Master's Thesis
Computer Science
Thesis no: MCS-2011-26
October 2011

Decoding algorithms of Reed-Solomon code

Szymon Czynszak

School of Computing
Blekinge Institute of Technology
SE – 371 79 Karlskrona
Sweden

Contact Information:
Author(s):
Szymon Czynszak
E-mail: szymon.czynszak@gmail.com

University advisor(s):
Mr Janusz Biernat, prof. PWR, dr hab. inż.
Politechnika Wrocławska
E-mail: janusz.biernat@pwr.wroc.pl

Mr Martin Boldt, dr
Blekinge Institute of Technology
E-mail: martin.boldt@bth.se

School of Computing
Blekinge Institute of Technology
SE – 371 79 Karlskrona
Sweden

Internet	: www.bth.se/com
Phone	: +46 455 38 50 00
Fax	: +46 455 38 50 57

Abstract

Reed-Solomon code is nowadays broadly used in many fields of data transmission. Using of error correction codes is divided into two main operations: information coding before sending information into communication channel and decoding received information at the other side. There are vast of decoding algorithms of Reed-Solomon codes, which have specific features. There is needed knowledge of features of algorithms to choose correct algorithm which satisfies requirements of system. There are evaluated cyclic decoding algorithm, Peterson-Gorenstein-Zierler algorithm, Berlekamp-Massey algorithm, Sugiyama algorithm with erasures and without erasures and Guruswami-Sudan algorithm. There was done implementation of algorithms in software and in hardware. Simulation of implemented algorithms was performed. Algorithms were evaluated and there were proposed methods to improve their work.

Contents

List of Figures

List of Tables

1. Introduction

With evolution of digital systems, the communication channels transport more and more data. This data may be affected by harmful factors like damping or interference. When this happens, then information which was transferred may be corrupted. To provide better performance for data transfer there can be used error-correction codes. Error-correction coding is a technique of adding redundance to transmitted data in order to detect and possibly repair damaged received data. Data must be encoded before sending into communication channel and decoded at the other end. There were devised several error-correction codes like Golay code, Goppa code, Turbo code, Hamming code, Reed-Solomon code etc. Main focus in the master thesis lies on decoding algorithms of Reed-Solomon code. Reed-Solomon code is widely used in CDs, DVDs, Blu-Ray, DSL, WiMAX or RAID 6.

Reed-Solomon code was invented in 1960 [1]. The $RS(n,k)$, where n denotes length of codeword, k denotes number of data symbols and there are $n-k$ control symbols is able theoretically to correct up to $t = \frac{n-k}{2}$ errors. Reed-Solomon code may be seen as non-binary BCH (Bose- Chaudhuri-Hocquenghem) code and particular decoding algorithms for BCH code can be used together for Reed-Solomon code. Reed-Solomon code is also cyclic code, so decoding algorithms for cyclic codes can be used. In 1960 [2] Peterson presented a method for decoding BCH codes and in 1961 [3] Gorenstein and Zierler tailored Peterson's method to Reed-Solomon code's purpose. In 1968 [4] Berlekamp presented an algorithm which was simplified by Massey in 1969 [5]. In 1975 [14] Sugiyama et al. invented another algorithm for decoding Reed-Solomon codewords. All these algorithms have in common that they employ error-locator polynomials to produce the result, but they do this step in a different way. They are called bounded distance decoding algorithms, because they produce one unique result.

In 1997 Sudan developed algorithm for decoding Reed-Solomon codewords, which was improved by Sudan and Guruswami in 1999 [7]. This algorithm, in contradistinction to bounded distance decoding algorithms, generates list of codewords within given range from decoded word. In 2003 Koetter and Vardy presented modification of Guruswami-Sudan algorithm to employ soft-decoding [19].

Algorithms are characterized by different computational complexity which is dependent on number of information and control elements in a codeword, degree of scalability, memory requirements, performance of processing, type of given output, suitability to use in multithreaded environment or vectorization. There are different environments where decoding algorithms may be used. Some of them are personal computers, industry computers, embedded systems, specific assembled hardware devices etc. Each of them are characterized by specific requirements and impose several contraints.

Implementation of decoding algorithms both in hardware and software may be accomplished in different ways. To construct decoder there can be used several mechanisms and algorithms which usually come from algebra domain. Algorithms are built with use of Galois field arithmetic which includes basic operations like addition, multiplication and calculation of field elements and operations which involve addition, multiplication, division and calculation of division's remainder for polynomials. There are vast of electronical elements, which may be used to construct hardware decoder.

However this is not simple task to build efficient decoder, when there are several ways to implement it. Furthermore, there are different types of algorithms, including bounded distance decoding algorithms, list decoding algorithms and soft decoding algorithms and there may be confusion to choose right one without knowing their characteristics of processing. Analysis of complexity and scalability is done to find out the main advantages and disadvantages of algorithms. There will be processed evaluation of presented solutions of implementation of algorithms.

1.1 Structure of document

Master thesis report is divided into several chapters. First is presented information about Galois fields. There are described what are Galois fields, how arithmetic operations work for them, how can be they represented in computer systems. Next chapter describes Reed-Solomon codes. There are given two definitions of Reed-Solomon code and is shown that these definitions generate the same code. Next chapter presents decoding algorithms. First is given introduction to basic decoding methods like complete decoder, standard array and syndrome decoding, then cyclic decoding, Peterson-Gorenstein-Zierler algorithm, Berlekamp-Massey algorithm, Sugiyama algorithm. There are described also auxiliary like Chien search and Forney algorithm. There is described modified Sugiyama algorithm which employs decoding with erasures. Last sections in the chapter presents Guruswami-Sudan algorithm. Guruswami-Sudan algorithm can employ Kötter algorithm for polynomial interpolation, Roth-Ruckenstein algorithm for polynomial factorization and Kötter-Vardy algorithm for transformation of soft information to hard information. Next chapter gives overview how algorithms were implemented both in software and hardware. Next, there are chapters which describe simulation environment, results from experiments and evaluation of results. Last chapter includes conclusions and future work topics.

1.2 Research questions

There are several decoding algorithms of Reed-Solomon code and they differ in their features like computional complexity and error-correction capability. Other factors which influence work of algorithms are number of occured error, codeword length, number of control elements etc. Constructing decoder, there can be defined several requirements like time of decoding, ability to decoding specific number of errors or complexity of algorithm. First research question which is used to solve this problem is:

1. What are the features of scalability and complexity of implemented algorithms?

Decoding algorithms usually consist of several steps. Some of these steps can be executed by different algorithms. Algorithms may be also modified in order to improve their work. Question which is used to find what can be done to improve algorithm work on algorithm level is:

2. What mathematical functions and modifications on algorithm level can improve performance of decoding algorithms?

Work of algorithms can be also improved by using features of software and hardware. Some steps of algorithms can be done by parallel units or for instance some precomputed data may be used to decrease time of decoding. Question which is used to find what can be done to improve work of algorithms on software and hardware level is:

3. Which features of software and hardware can improve performance of decoding algorithms?

1.3 Aims and objectives

Aim of the master thesis is to evaluate implemented decoders of Reed-Solomon code under criteria of scalability and complexity.

There are following objectives which are fullfiled in order to answer research questions:

- preparation of mathematical functions which concern mainly Galois fields, polynomial arithmetic, matrices and vectors.
- implementation of algorithms in software and hardware,
- finding out improvements for algorithms,
- simulation of proposed solutions in simulator,

- evaluation of proposed solutions in terms of complexity and scalability.

1.4 Research methodology

To get overview of current knowledge in decoding algorithms of Reed-Solomon code area, background study is done. Main research method used in the master thesis is an experiment. To get done experiment, algorithms from given set must be implemented. Set consists of Peterson-Gorenstein-Zierler algorithm, Berlekamp-Massey algorithm, Sugiyama algorithm, Guruswami-Sudan algorithm. Hard-decision algorithms which include Peterson-Gorenstein-Zierler algorithm, Berlekamp-Massey algorithm, Sugiyama algorithm, Guruswami-Sudan algorithm are popular methods for decoding Reed-Solomon words [26] and Koetter-Vardy algorithm is a popular soft-decision algorithm [21]. That was the criterion to complete the set. There is created list of criteria under which algorithms and decoders will be compared. Implementation is done in C++ programming language. First are implemented basic mathematical functions which include arithmetic of Galois field, arithmetic of polynomials, functions needed for operations with matrices and vectors. After this step, implementation of selected decoding algorithms is done. To provide useful information how algorithms work in software, simulator is created. Simulator allows to define Galois fields of characteristic 2 and of chosen exstension, create Reed-Solomon code of given number of information words and control words, choose number of random selected errors. The other advantage of simulator is that it can be used to validate correctness of implemented algorithms. Simulator may gather information like time of decoding of received word, how much memory was used during decoding process, whether received word was correctly decoded. Implementation of decoding algorithms in hardware is preceded by design of decoder. The most popular decoding algorithms used in hardware employ error locator polynomials to compute the result of decoding [27]. Peterson-Gorenstein-Zierler algorithm, Berlekamp-Massey algorithm, Sugiyama algorithm use error locator polynomials during decoding operation. There must be chosen which electronic elements can be used and how to solve construction of decoder. Implementation is done in VHDL language. Correctness of developed algorithms is checked in ISim simulator which provides valuable information about time of decoding, result of decoding. Evaluation of decoding algorithms is mainly based on results delivered in simulation process conducted in software and information gathered from simulator ISim.
Independent variables of experiment are:

- length of codeword,

- number of control elements,
- number of occured errors,
- noise in the communication channel,
- number of threads.

Dependent variables are:

- time of decoding,
- memory required,
- status of decoding (no errors, corrected, not correctable),
- quality of decoding (correctness of decoding, number of results in list decoding),
- complexity of decoder (number of executed loops, number of electronic elements in circuit).

To evaluate scalability of algorithms, independent variables of experiment are manipulated. Algorithms' processing may (but not must) vary with different values of number of errors, length of codeword, number of control elements etc. Threads may be used to lessen processing time of decoding. Number of control elements and length of codeword may affect complexity of hardware decoder, because mostly of length of registers. As a result of experiment there will be given an answer how these independent variables affect dependent variables. Simulator is used to provide information about processing of algorithms. Gathered data will be evaluated in relation to scalability and complexity.

Threats to internal validity:

- implementation of algorithms is affected by programming skills of programmer, so the theoretical decoding process may be different than real life decoding process. That's why there is not compared exact time of decoding between different algorithms, but complexity - how working of algorithms is affected by independent variables.
- there can occur confounding variables during testing, for example exchanging content of cache memory can affect working of big data structures, while it is not problem for small data structures.

Threats to external validity:

- there is evaluated set of Reed-Solomon codes, but not all codes are included in this set, because there is no time to evaluate them all. Codes to evaluate are chosen so, that there are representatives from Reed-Solomon codes of length 7, 15, 31, 63. Using longer codes takes too much time to evaluate them.
- algorithms are implemented on x86 machine and they use specific features of this hardware. However, there are no tested on other machines, whose features may affect decoding process.

- disrupting of received vector from communication channel is occured by noise. The method used for noise generation is designed by author of master thesis. However there are some “standard” channel models like AWGN (Additive White Gaussian Noise) etc., where decoding algorithm can work differently.

2. Galois fields

2.1 Introduction

Reed-Solomon codes use as code symbols elements from extended fields which are also known as extended Galois fields [9]. Galois field is a finite set of elements with defined operations of addition and multiplication, which has following properties:

- result of adding or multiplying two elements in Galois field is an element in the same Galois field,
- identity element of addition is 0, each element in Galois field has additive inverse,
- identity element of multiplication is 1, each element in Galois field has multiplicative inverse,
- addition and multiplication are commutative and associative,
- multiplication distributes over addition.

Galois field which consists of q elements is denoted as $GF(q)$. Number of elements of $GF(q)$ is p^m, where p is a prime number which is called characteristic of field, and m is called extension order. Field A is extension of field B, if field B is a subfield of field A. Subfield of field A is field B, if each element of field B lies in the field A and properties of the field B are satisfied for operations of addition and multiplication which come from field A.

Following set describes elements in $GF(q)$:

$$\{0, \alpha^0, \alpha^1, \ldots, \alpha^{q-2}\} \equiv \{0, 1, \alpha, \ldots, \alpha^{q-2}\}$$

Element α in $GF(p^m)$ is the root of primitive polynomial $w(x)$ of degree m with coefficients from field $GF(p)$. A primitve polynomial $w(x)$ is irreducible and divides polynomial x^n+1, where $n = p^m-1$, and does not divide any polynomial $x^z + 1$, where $z < n$. Polynomial $w(x)$ of degree m is irreducible, if it is not divided by any polynomial of degree z, where $0 < z < m$. Roots of polynomial $w(x)$ of degree m with coefficients from $GF(p)$ are all primitive elements of $GF(p^m)$. Primitive elements of $GF(p^m)$ are defined as elements, which has multiplicative order equal to $p^m - 1$. Multiplicative order of element α^i is such natural number r, which satisfies $(\alpha^i)^r = 1$ for $0 \leq i \leq q-2$. Reciprocal polynomials of primitive polynomials are also primitive polynomials. Reciprocal polynomial for polynomial $w(x)$ of degree m satisfies:

$$w^*(x) = \tfrac{x^m}{w_0} w(\tfrac{1}{x})$$

If reciprocal polynomial is identical to original polynomial, then such polynomial is called self-reciprocal.

2.2 Representation of elements in $GF(2^m)$

Reed-Solomon codes are often based on Galois field of characteristic 2, because elements in $GF(2^m)$ can be expressed as binary vectors, and arithmetic units for binary elements in $GF(2)$ can be used.

2.2.1 Polynomial representation

Elements in $GF(2^m)$ can be expressed as [9]:

$$\alpha^i \rightarrow R^{(\alpha^i)}(x) = x^i \bmod p(x) \ ,$$

where $p(x)$ is primitive polynomial over $GF(2^m)$. Element α^i in $GF(2^m)$ can be expressed as m-dimensional vector of binary coefficients of polynomial $R^{(\alpha^i)}(x) \rightarrow (R_0^{(\alpha^i)}, R_1^{(\alpha^i)}, \ldots, R_{m-2}^{(\alpha^i)}, R_{m-1}^{(\alpha^i)})$. This method is often used in digital technology. Element $0 \rightarrow R^{(0)}(x)$ is represented as m-dimensional vector filled with zeroes.

Addition

Adding two elements in $GF(2^m)$ in polynomial representation can be done with XOR function:

$$\alpha^i + \alpha^j \rightarrow \sum_{n=0}^{m-1} (R_n^{(\alpha^i)} \oplus R_n^{(\alpha^j)}) x^n$$

Furthermore:

$$0 + \alpha^i = \alpha^i + 0 = \alpha^i$$

Choice of primitive polynomial affects addition operations $GF(2^m)$.

Example 2.1.
Field $GF(8)$ can be created by primitive polynomials:
$p_1(x) = x^3 + x + 1$ and $p_2(x) = x^3 + x^2 + 1$. Polynomial $p_2(x)$ is reciprocal polynomial of $p_1(x)$. For field created by $p_1(x)$ is true that $\alpha^3 = \alpha + 1$, and for field created by $p_2(x)$ is true that $\alpha^3 = \alpha^2 + 1$.

Subtraction

Subtraction is equivalent to addition in fields of characterstic 2.

$$\alpha^i + \alpha^j = \alpha^i - \alpha^j$$

Multiplication

Multiplication of two elements in $GF(2^m)$ for polynomial representation is done as follows:

Furthermore, true is that:

$$0 \cdot \alpha^i = \alpha^i \cdot 0 = 0$$

Division

Division of two elements in $GF(2^m)$ for polynomial representation is done as follows:

$$\frac{\alpha^i}{\alpha^j} \to \alpha^i \cdot \alpha^{-j}$$

Additive inverse

Each element in $GF(2^m)$ is also own additive inverse:

$$\alpha^i = -\alpha^i$$

Multiplicative inverse

In $GF(2^m)$ multiplicative inverse can be expressed as:

$$\alpha^{-i} = \begin{cases} \alpha^{2^m-1-i} & \text{for } 1 \leq i \leq 2^m - 2, \\ 1 & \text{for } i = 0. \end{cases}$$

Element 0 doesn't have multiplicative inverse.

2.2.2 Positive integer representation

Elements in $GF(2^m)$ can be expressed as [9]:

$$\alpha^i \to N^{(\alpha^i)} = \log_\alpha(\alpha^i) + 1 = i + 1$$

Element 0 is express as $N^{(0)} = 0$.

Addition

Addition of two elements in $GF(2^m)$ in positive integer representation is as follows:

$$\alpha^i + \alpha^j \to \begin{cases} (N^{(\alpha^j)} + Z(N^{(\alpha^i)} - N^{(\alpha^j)}) - 1) \bmod (2^m - 1) + 1 & \text{for } i > j, \\ 0 & \text{for } i = j. \end{cases}$$

$Z(N^{(\alpha^i)})$ denotes Zech logarithm for element α^i.

Furthermore, true is that:

$$0 + \alpha^i = \alpha^i + 0 = \alpha^i$$

Multiplication

Multiplication of two elements in $GF(2^m)$ in positive integer representation is as follows:

$$\alpha^i \cdot \alpha^j \to (N^{(\alpha^i)} + N^{(\alpha^j)} - 2) \bmod (2^m - 1) + 1$$

Furthermore, true is that:

$$0 \cdot \alpha^i = \alpha^i \cdot 0 = 0$$

Principles for subtraction, division, additive inverse and multiplicative inverse are the same as for polynomial representation.

2.2.3 Vector representation

There is given following primitive polynomial: Dany jest wielomian pierwotny

$$p(x) = p_m x^m + p_{m-1} x^{m-1} + \ldots + p_1 x + p_0,$$

which can be used for creation of $GF(2^m)$. With this primitive polynomial generated is periodic sequence, which can be used for representation of elements in $GF(2^m)$ [9]. Equation for $j+m$'s element s_{j+m} of periodic sequence is as follows:

$$s_{j+m} = s_{j+m-1} p_{j+m-1} + s_{j+m-2} p_{j+m-2} + \ldots + s_j p_j$$

At the beginning of algorithm there must be stated which element initializes sequence. During writing this master thesis, following elements were used:$s_0 = 1$, $s_1 = 0$, $s_2 = 0$, $\ldots$, $s_{m-1} = 0$.

Representation of elements in $GF(2^m)$ is as follows:

- element 0 is denoted as m-dimensional vector of zeroes – $\underbrace{(0, 0, \ldots, 0)}_{m}$,
- element α^i for $0 \leq i \leq 2^m - 2$ is denoted as following vector – $(s_i, s_{i+1}, s_{i+2}, \ldots, s_{i+m-1})$.

Addition in vector representation is done the same as for polynomial representation, but method for multiplication for polynomial representation cannot be used for vector representation.

2.3 Zech logarithms

Zech logarithms can be used for addition of elements in $GF(2^m)$ in logarithmic domain [9]. Zech logarithm is defined as:

$$\alpha^{Z(x)} = 1 + \alpha^x \qquad Z(x) = \log_\alpha(1 + \alpha^x)$$

Addition of two elements in $GF(2^m)$ can be done as follows:

$$\alpha^i + \alpha^j = \alpha^i(1 + \alpha^{j-i}) = \alpha^i \alpha^{Z(j-i)} = \alpha^{i+Z(j-i)}$$

For $GF(2^m)$ it is assumed that $Z(0) = -\infty$ and $Z(-\infty) = 0$.
For $GF(2^m)$ it is true that:

$$(Z(x) - x)2^i \bmod (2^m - 1) = Z((2^m - 1 - x)2^i \bmod (2^m - 1)) \tag{2.1}$$

$$Z(x)2^i \bmod (2^m - 1) = Z(2^i x \bmod (2^m - 1)) \tag{2.2}$$

Using equations 2.1 and 2.2 there can be created table of Zech logarithms.

Example 2.2.
There will be computed table of Zech logarithms for field $GF(8)$ created by polynomial $p(x) = x^3 + x + 1$. It is true that:

$$\alpha^3 = \alpha + 1 \qquad \text{and} \qquad \alpha^{Z(x)} = \alpha^x + 1,$$

so $Z(1) = 3$. Using equations 2.1 i 2.2 and assigning $m = 3$, $x = 1$, there can be computed next Zech logarithms.
Equation 2.1:

$$\text{dla } i = 0 \qquad \begin{aligned} (Z(1) - 1)2^0 \bmod (2^3 - 1) &= Z((2^3 - 1 - 1)2^0 \bmod (2^3 - 1)) \\ (3 - 1)1 \bmod 7 &= Z(6 \cdot 1 \bmod 7) \\ 2 \bmod 7 &= Z(6 \bmod 7) \\ 2 &= Z(6) \end{aligned}$$

$$\begin{aligned}
\text{dla } i=1 \qquad (Z(1)-1)2^1 \bmod (2^3-1) &= Z((2^3-1-1)2^1 \bmod (2^3-1)) \\
(3-1)2 \bmod 7 &= Z(6\cdot 2 \bmod 7) \\
4 \bmod 7 &= Z(12 \bmod 7) \\
4 &= Z(5)
\end{aligned}$$

$$\begin{aligned}
\text{dla } i=2 \qquad (Z(1)-1)2^2 \bmod (2^3-1) &= Z((2^3-1-1)2^2 \bmod (2^3-1)) \\
(3-1)4 \bmod 7 &= Z(6\cdot 4 \bmod 7) \\
8 \bmod 7 &= Z(24 \bmod 7) \\
1 &= Z(3)
\end{aligned}$$

Equation 2.2:

$$\begin{aligned}
\text{dla } i=1 \qquad Z(1)2^1 \bmod (2^3-1) &= Z(2^1\cdot 1 \bmod (2^3-1)) \\
3\cdot 2 \bmod 7 &= Z(2 \bmod 7) \\
6 &= Z(2)
\end{aligned}$$

$$\begin{aligned}
\text{dla } i=2 \qquad Z(1)2^2 \bmod (2^3-1) &= Z(2^2\cdot 1 \bmod (2^3-1)) \\
3\cdot 4 \bmod 7 &= Z(4 \bmod 7) \\
5 &= Z(4)
\end{aligned}$$

$$\begin{array}{llll}
Z(-\infty)=0 & Z(0)=-\infty & Z(1)=3 & Z(2)=6 \\
Z(3)=1 & Z(4)=5 & Z(5)=4 & Z(6)=2
\end{array}$$

2.3.1 Imamura algorithm

Imamura algorithm is an iterative algorithm for computing Zech logarithms, which can be easily implemented programmatically [16].

Imamura algorithm is as follows ($\log_\alpha(0)$ doesn't exist and is denoted as $\lim\limits_{x\to 0}\log_\alpha x = -\infty$):

1. Each element of $GF(2^m)$ express as polynomial $R^{(i)}(x)$, where $i \in \{0, 1, \alpha, \ldots, \alpha^{2^m-2}\}$.

$+$	0	1	α	α^2	α^3	α^4	α^5	α^6
0	0	1	α	α^2	α^3	α^4	α^5	α^6
1	1	0	α^3	α^6	α	α^5	α^4	α^2
α	α	α^3	0	α^4	1	α^2	α^6	α^5
α^2	α^2	α^6	α^4	0	α^5	α	α^3	1
α^3	α^3	α	1	α^5	0	α^6	α^2	α^4
α^4	α^4	α^5	α^2	α	α^6	0	1	α^3
α^5	α^5	α^4	α^6	α^3	α^2	1	0	α
α^6	α^6	α^2	α^5	1	α^4	α^3	α	0

Table 2.1: Addition table for elements in $GF(8)$ created by polynomial $p(x) = x^3 + x + 1$.

2. For each polynomial $R^{(i)}(x)$ compute $N(\log_\alpha(i)) = R^{(i)}(2)$ in the field of integer numbers.
3. Assign:

$$N(Z(\log_\alpha(i))) = \begin{cases} N(\log_\alpha(i)) - 1 & \text{for } N(\log_\alpha(i)) \text{ odd,} \\ N(\log_\alpha(i)) + 1 & \text{for } N(\log_\alpha(i)) \text{ even.} \end{cases}$$

4. With values $N(\log_\alpha(i))$ and $N(Z(\log_\alpha(i)))$, find $Z(\log_\alpha(i))$.

Example 2.3.
Example of calculation of Zech logarithms for field $GF(8)$ created by polynomial $p(x) = x^3 + x + 1$:

$\log_\alpha(i)$	i	$R^{(i)}(x)$	$N(\log_\alpha(i))$	$N(Z(\log_\alpha(i)))$	$Z(\log_\alpha(i))$
$-\infty$	0	0	0	1	0
0	1	1	1	0	$-\infty$
1	α	x	2	3	3
2	α^2	x^2	4	5	6
3	α^3	$x+1$	3	2	1
4	α^4	x^2+x	6	7	5
5	α^5	x^2+x+1	7	6	4
6	α^6	x^2+1	5	4	2

Results are the same as for example 2.2.

$\cdot$	0	1	α	α^2	α^3	α^4	α^5	α^6
0	0	0	0	0	0	0	0	0
1	0	1	α	α^2	α^3	α^4	α^5	α^6
α	0	α	α^2	α^3	α^4	α^5	α^6	1
α^2	0	α^2	α^3	α^4	α^5	α^6	1	α
α^3	0	α^3	α^4	α^5	α^6	1	α	α^2
α^4	0	α^4	α^5	α^6	1	α	α^2	α^3
α^5	0	α^5	α^6	1	α	α^2	α^3	α^4
α^6	0	α^6	1	α	α^2	α^3	α^4	α^5

Table 2.2: Multiplication table for elements in $GF(8)$ created by polynomial $p(x) = x^3 + x + 1$.

$+$	0	1	$\cdots$	α^{2^m-3}	α^{2^m-2}
0	0	1	$\cdots$	α^{2^m-3}	α^{2^m-2}
1	1	0	$\cdots$	$\alpha^{2^m-3} + 1$	$\alpha^{2^m-2} + 1$
$\vdots$	$\vdots$	$\vdots$	0	$\vdots$	$\vdots$
α^{2^m-3}	α^{2^m-3}	$1 + \alpha^{2^m-3}$	$\cdots$	0	$\alpha^{2^m-2} + \alpha^{2^m-3}$
α^{2^m-2}	α^{2^m-2}	$1 + \alpha^{2^m-2}$	$\cdots$	$\alpha^{2^m-3} + \alpha^{2^m-2}$	0

Table 2.3: Addition table for elements in $GF(2^m)$.

$\cdot$	0	1	$\cdots$	α^{2^m-3}	α^{2^m-2}
0	0	0	0	0	0
1	0	1	$\cdots$	α^{2^m-3}	α^{2^m-2}
$\vdots$	0	$\vdots$	$\ddots$	$\vdots$	$\vdots$
α^{2^m-3}	0	α^{2^m-3}	$\cdots$	$\alpha^{2^m-3} \cdot \alpha^{2^m-3}$	$\alpha^{2^m-2} \cdot \alpha^{2^m-3}$
α^{2^m-2}	0	α^{2^m-2}	$\cdots$	$\alpha^{2^m-3} \cdot \alpha^{2^m-2}$	$\alpha^{2^m-2} \cdot \alpha^{2^m-2}$

Table 2.4: Multiplication table for elements in $GF(2^m)$.

2.4 LUT tables

2.4.1 Addition

Addition table for elements in $GF(2^m)$ is shown in table 2.3.
Looking at addition table, it can be noticed that:

- if one of element in addition is 0, then the result of addition will be the second element, so from table there can be excluded $2 \cdot 2^m - 1$ elements (results of operations, where one of components is 0),
- if added are the same elements, then result will be 0, so from table there can be excluded 2^m elements (diagonal of zero elements),
- table is symmetric for diagonal of zeroes, so there can be excluded $\frac{2^{2m}}{2} - \frac{2^m}{2}$ elements (half of all elements minus half of elements on diagonal of zeroes)/

Elements, which can be excluded are marked with orange colour in table 2.3. After calculation, there must be stored following number of elements in addition LUT:

$$N_{LUT+} = 2^{2m-1} - 3 \cdot 2^{m-1} + 1$$

Addition algorithm with use of LUT table is as follows:

1. Return 0, if added are the same elements, else continue.
2. If one component in addition is 0, then return second component, else continue.
3. For $\alpha^i + \alpha^j$ and $i > j$, return value from cell $[\alpha^i][\alpha^j]$, where first parameter is number of column of LUT table, and second parameter is number of row in LUT table, else for $i < j$ return result from cell $[\alpha^j][\alpha^i]$.

2.4.2 Multiplication

Multiplication table for elements in $GF(2^m)$ is shown in table 2.4. Similarly like in addition table, multiplication table is symmetric.

There can be noticed following properties of multiplication table:

- if one of components in multiplication is zero, then result will be also zero, so from table can be excluded $2 \cdot 2^m - 1$ elements (results of operations, where one of components is 0),
- if one of components is 1, then result will be second component, so from table can be excluded $2 \cdot 2^m - 1$ elements (result of operations, where one of components is 1),
- table is symmetric for diagonal of products of the same elements, so from table can be excluded $\frac{2^{2m}}{2} - \frac{2^m}{2}$ elements (half of all elements minus half of elements on diagonal of results of the same elements).

Elements which can be excluded from table are marked with orange colour in table 2.4. After calculcation, number of elements which must be stored in LUT table is as follows:

$$N_{LUT\cdot} = 2^{2m-1} - 3 \cdot 2^{m-1} + 1$$

Multiplication algorithm with use of LUT table is as follows:

1. Return element 0, if one of multiplication components is 0, else conitnue.
2. If one of multiplication components is 1, then return second component, else continue.
3. For $\alpha^i + \alpha^j$ and $i \geq j$, return result from cell $[\alpha^i][\alpha^j]$, where first parameter is number of column in LUT, and second parameter is number of row in LUT, else for $i < j$ return result from cell $[\alpha^j][\alpha^i]$.

Exemplary addition and multiplication tables for $GF(8)$ created by polynomial $p(x) = x^3 + x + 1$ are presented respectively in table 2.1 and 2.2.

3. Reed-Solomon code

3.1 Introduction

Reed-Solomon code is error-correction code [9] [1] [12] [14]. Error-correction is such code, which can be used for error correction. Furthermore, Reed-Solomon code is:

- linear code, where result of linear combination of any two codewords is codeword,
- cyclic code, where result of cyclic shift of codeword is codeword,
- block code, where data sequence is divided into blocks of fixed lengths and these blocks are coded independently,
- nonbinary code, where code symbols are not binary - 1 nor 0.

Codeword is ordered set of code symbols and belongs to code. Code symbols are elements in $GF(q)$, which are used to code construction.

Reed-Solomon, where codeword is of length n elements and encodes k information elements, is denoted as $RS(n, k)$. Code $RS(n, k)$ has following properties:

- codeword consists of n code symbols,
- codeword encodes k information elements,
- number of redundant control elements is $r = n - k$,
- code has correction capability equals to $t = \lfloor \frac{r}{2} \rfloor$,
- minimal Hamming distance is $d_{min} = r + 1$.

Code can correct all error patterns, where number of errors is less or equal error capability t. Minimal Hamming distance d_{min} determines minimal number of positions, where two vectors are different. If code is created over $GF(2^m)$, then code is of form $RS(2^m - 1, 2^m - 1 - 2t)$. Reed-Solomon codes satifies Singleton bound with equality– $d_{min} \leq n - k + 1$, which means, that has the best error-correction capability with given n and given information elements k [6]. Codes which satisfy Singleton bound with equality are called MDS codes - Maximum Distance Separable.

3.2 Encoding

Encoding is process of transformation information elements into codeword. There are presented two methods for creation of Reed-Solomon codeword.

3.2.1 Original method

Given are information elements $m_i \in GF(2^m)$ for $0 \leq i \leq k-1$, which are used for codeword construction [1] [7]. Given is polynomial:

$$m(x) = m_{k-1}x^{k-1} + m_{k-2}x^{k-2} + \ldots + m_1 x + m_0 \tag{3.1}$$

Given is set of different non-zero elements in $GF(q)$ which is called support set [7]:

$$(\alpha_1, \alpha_2, \ldots, \alpha_n)$$

Codeword of $RS(n,k)$ code can be expressed as:

$$c = (m(\alpha_1), m(\alpha_2), \ldots, m(\alpha_n))$$

During work on master thesis, used support set was:

$$(1, \alpha, \alpha^2, \ldots, \alpha^{n-1}) \tag{3.2}$$

Codeword of $RS(n,k)$ code created with support set (3.2) can be expressed as:

$$c = (m(1), m(\alpha), \ldots, m(\alpha^{n-1})) \tag{3.3}$$

or

$$c(x) = m(\alpha^{n-1})x^{n-1} + m(\alpha^{n-2})x^{n-2} + \ldots + m(\alpha)x + m(1) \tag{3.4}$$

For codeword of $RS(n,k)$ code created with support set (3.2) satisfied is that:

$$c(\alpha) = c(\alpha^2) = \ldots = c(\alpha^{2t}) = 0$$

Equation for Galois Field Fourier Transform for polynomial $f(x)$ of degree $n-1$ is as follows [8]:

$$F(x) = GFFT(f(x)) = f(1) + f(\alpha)x + f(\alpha^2)x^2 + \ldots + f(\alpha^{n-1})x^{n-1}$$

With use of this equation, there can be shown for codeword $c(x)$ of $RS(n,k)$ code created with support set (3.2) that $2t$ next coefficients are equal to zero. Codeword $c(x)$ is in encoded in time domain, and $C(x)$ is encoded in frequency domain [14].

Example 3.1.
Given is following vector of information elements:

$$m = (\alpha, \alpha^2, 1) \rightarrow m(x) = \alpha + \alpha^2 x + x^2$$

Codeword c of $RS(7,3)$ code for this vector is as follows:

$$\begin{aligned} c &= (m(1), m(\alpha), m(\alpha^2), m(\alpha^3), m(\alpha^4), m(\alpha^5), m(\alpha^6)) \\ &= (\alpha^5, \alpha^6, \alpha, 0, \alpha^6, 0, \alpha^5) \end{aligned}$$

Codeword

$$c \to c(x) = \alpha^5 + \alpha^6 x + \alpha x^2 + \alpha^6 x^4 + \alpha^5 x^6$$

transformed with Galois Field Fourier Transform is as follows:

$$\begin{aligned} C &= (c(1), c(\alpha), c(\alpha^2), c(\alpha^3), c(\alpha^4), c(\alpha^5), c(\alpha^6)) \\ &= (\alpha, 0, 0, 0, 0, 1, \alpha^2) \end{aligned}$$

Consecutive $2t$ positions in C are zeroes.

3.2.2 RS code as nonbinary BCH code

Reed-Solomon codes can be seen as nonbinary BCH codes [10] [9]. For construction of codeword of $RS(n,k)$ code can be used following generator polynomial:

$$g(x) = (x - \alpha)(x - \alpha^2)\dots(x - \alpha^{2t}) \qquad \text{where } \alpha^i \in GF(n+1)$$

Given is information polynomial $m(x)$ 3.1. Codeword of nonsystematic code $RS(n,k)$ is as follows:

$$c(x) = m(x)g(x)$$

Codeword of systematic $RS(n,k)$ code is as follows:

$$c(x) = m(x)x^{2t} + Q(x) \qquad \text{where } Q(x) = m(x)x^{2t} \bmod g(x)$$

$RS(n,k)$ code can be created with use of original method and as nonbinary BCH code. Both of there codes satisfy [8]:

- code created with both definitions creates the same linear space,
- code created with both definitions is cyclic,
- for codeword created by both definitions satisfied is $c(\alpha_1) = c(\alpha_2) = \ldots = c(\alpha_{2t}) = 0$ for some $2t$ consecutive elements in $GF(q)$.

4. Decoding algorithms

4.1 Introduction

Decoding is a process of transformation of received vector from communication channel onto sequence of information elements which were used to create codeword. It may happen that received vector is not codeword, because codeword was corrupted in communication channel. In this case code can correct number of errors up to value of error correction capability. If decoder is not able to correct received vector and such situation can be recognized then it is called decoder failure [15]. If there occured so many errors that received vector is decoded onto different codeword than original, then such situation is called decoder error.

4.1.1 Complete decoder

Complete decoder works as follows [12]:

1. Compute Hamming distance between received vector and each codeword of $RS(n, k)$ code,
2. Choose codeword, where value of Hamming distance was least.

Complete decoder is impractical, because there are q^k of codewords of $RS(n, k)$ code and it would take much time to find result.

4.1.2 Standard array

Standard array is such matrix of $RS(n, k)$ which consists of q^k columns and q^{n-k} rows [10]. In first rows there are written all q^k codewords of $RS(n, k)$ code, and in last rows vectors which are close to them within Hamming distance. Decoding is a process of finding received vector in standard array and result of decoding is codeword in first row of the same column where received vector was found. This method is also impractical, because there must be stored q^n vectors, which may be huge number.

4.1.3 Syndrome decoding

$RS(n, k)$ code is a k-dimensional subspace of n-dimensional space. It is connected with two following matrices [10] [9]:

- generator matrix G, which consists of k linear independent vectors, which are used to create codeword – $c = mG$, where m denotes k-dimensional vector of information elements,

- parity check matrix H, which is used to decode received vector.

Parity check matrix is used to compute syndromes. Syndrome s is computed as follows:

$$s = rH^T,$$

where r denotes received vector, and H^T is transpozed matrix H. If received vector is codeword, then syndrome is zero-value vector – $s = cH^T = 0$, else there are errors, which can be written that to codeword in communication channel was added error vector – $r = c + e$, where e denotes error vector.

For $RS(n, k)$ code generator matrix is as follows:

$$G = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_k \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & (\alpha)^2 & (\alpha^2)^2 & \dots & (\alpha^{n-1})^2 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & (\alpha)^{k-1} & (\alpha^2)^{k-1} & \dots & (\alpha^{n-1})^{k-1} \end{bmatrix}$$

Codeword for information elements $m = (m_1, \dots, m_k)$ can be expressed as:

$$c = mG = m_1g_1 + m_2g_2 + \dots + m_kg_k$$

Example 4.1.
Given is $RS(7, 3)$ code created in $GF(8)$ by polynomial $p(x) = x^3 + x + 1$. Generator matrix G is as follows:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \end{bmatrix}$$

For $m = (\alpha, \alpha^2, 1)$:

$$\begin{aligned} c = mG = (\alpha, \alpha^2, 1) &\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \end{bmatrix} \\ &= (\alpha, \alpha, \alpha, \alpha, \alpha, \alpha, \alpha) + (\alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, 1, \alpha) + (1, \alpha^2, \alpha^4, \alpha^6, \alpha, \alpha^3, \alpha^5) \\ &= (\alpha^5, \alpha^6, \alpha, 0, \alpha^6, 0, \alpha^5) \end{aligned}$$

Construction of codeword with matrix G gives the same results as to use original method. Let $m \rightarrow m(x)$:

$$(\alpha, \alpha^2, 1) \rightarrow \alpha + \alpha^2x + x^2$$

Using original method of encoding 3.3:

$$\begin{aligned} c &= (m(1), m(\alpha), m(\alpha^2), m(\alpha^3), m(\alpha^4), m(\alpha^5), m(\alpha^6)) \\ &= (\alpha^5, \alpha^6, \alpha, 0, \alpha^6, 0, \alpha^5) \end{aligned}$$

Parity check matrix for $RS(n,k)$ code is as follows:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & (\alpha)^2 & (\alpha^2)^2 & \ldots & (\alpha^{n-1})^2 \\ 1 & (\alpha)^3 & (\alpha^2)^3 & \ldots & (\alpha^{n-1})^3 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & (\alpha)^{2t} & (\alpha^2)^{2t} & \ldots & (\alpha^{n-1})^{2t} \end{bmatrix}$$

Let received vector be $r = (r_0, r_1, r_2, \ldots, r_{n-1})$:

$$\begin{aligned} s = rH^T &= (r_0, r_1, r_2, \ldots, r_{n-1}) \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & (\alpha)^2 & (\alpha^2)^2 & \ldots & (\alpha^{n-1})^2 \\ 1 & (\alpha)^3 & (\alpha^2)^3 & \ldots & (\alpha^{n-1})^3 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & (\alpha)^{2t} & (\alpha^2)^{2t} & \ldots & (\alpha^{n-1})^{2t} \end{bmatrix}^T \\ &= (r_0, r_1, r_2, \ldots, r_{n-1}) \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ \alpha & (\alpha)^2 & (\alpha)^3 & \ldots & (\alpha)^{2t} \\ \alpha^2 & (\alpha^2)^2 & (\alpha^2)^3 & \ldots & (\alpha^2)^{2t} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ \alpha^{n-1} & (\alpha^{n-1})^2 & (\alpha^{n-1})^3 & \ldots & (\alpha^{n-1})^{2t} \end{bmatrix} \\ &= (r_0, r_0, r_0, \ldots, r_0) + (r_1\alpha, r_1(\alpha)^2, r_1(\alpha)^3, \ldots, r_1(\alpha)^{2t}) + \ldots \\ &\quad + (r_{n-1}\alpha^{n-1}, r_{n-1}(\alpha^{n-1})^2, r_{n-1}(\alpha^{n-1})^3, \ldots, r_{n-1}(\alpha^{n-1})^{2t}) \\ &= (r_0 + r_1\alpha + r_2\alpha^2 + \ldots + r_{n-1}\alpha^{n-1}, \\ &\quad r_0 + r_1(\alpha)^2 + r_2(\alpha^2)^2 + \ldots + r_{n-1}(\alpha^{n-1})^2, \ldots, \\ &\quad r_0 + r_1(\alpha)^{2t} + r_2(\alpha^2)^{2t} + \ldots + r_{n-1}(\alpha^{n-1})^{2t}) \\ &= (\sum_{i=0}^{n-1} r_i\alpha^i, \sum_{i=0}^{n-1} r_i(\alpha^i)^2, \ldots, \sum_{i=0}^{n-1} r_i(\alpha^i)^{2t}) \\ &= (\sum_{i=0}^{n-1} r_i\alpha^i, \sum_{i=0}^{n-1} r_i(\alpha^2)^i, \ldots, \sum_{i=0}^{n-1} r_i(\alpha^{2t})^i) \end{aligned}$$

It can be noticed, that computing particular syndrome elements is computing value of polynomial $r(x)$, where

$$r(x) = r_{n-1}x^{n-1} + r_{n-2}x^{n-2} + \ldots + r_1x + r_0$$

For $RS(n,k)$ code, where roots of generator polynomial $g(x)$ are consecutive $2t$ elements $\alpha, \alpha^2, \ldots, \alpha^{2t}$, it is true that:

$$s = (r(\alpha), r(\alpha^2), \ldots, r(\alpha^{2t})) \tag{4.1}$$

Particular elements of syndrome vector are:

$$\begin{aligned} s_1 &= r(\alpha) \\ s_2 &= r(\alpha^2) \\ &\ldots \\ s_{2t} &= r(\alpha^{2t}) \end{aligned}$$

Example 4.2.
Parity check matrix for $RS(7,3)$ code:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha & \alpha^3 & \alpha^5 \\ 1 & \alpha^3 & \alpha^6 & \alpha^2 & \alpha^5 & \alpha & \alpha^4 \\ 1 & \alpha^4 & \alpha & \alpha^5 & \alpha^2 & \alpha^6 & \alpha^3 \end{bmatrix}$$

It can be written that:

$$s = rH^T = (c+e)H^T = cH^T + eH^T = 0 + eH^T = eH^T$$

Syndrome is then dependent only from error vector, which means, that there can be created array, where syndrome vectors are assigned to error patterns. Such array has 2 columns and q^{n-k} rows for $RS(n,k)$ code in $GF(q)$. Decoding is a process of computing syndrome and then finding error pattern in array which is assigned to computed syndrome. When error pattern is found, then it is added to received vector. This method is still impractical, because size of such array may be huge.

Example 4.3.
There are given three codewords of $RS(7,3)$ code created over $GF(8)$ with generator polynomial $p(x) = x^3 + x + 1$:

- $c_1(x) = x^6 + \alpha^5 x^5 + \alpha^2 x^4 + \alpha x^2 + \alpha^3 x + \alpha^4$,
- $c_2(x) = x^6 + \alpha^2 x^5 + \alpha x^4 + \alpha^4 x^3 + \alpha^5 x^2 + \alpha^3$,
- $c_3(x) = \alpha^5 x^6 + \alpha^6 x^4 + \alpha x^2 + \alpha^6 x + \alpha^5$.

All three vectors are results of decoding information vector $m = (\alpha, \alpha^2, 1)$ with three methods: as nonsystematic BCH code, as systematic BCH code, with original method. All three vectors lies in the same linear space of $RS(7,3)$ code. Let error vector be $e(x) = \alpha x^6 \rightarrow e = (0,0,0,0,0,0,\alpha)$. Received vectors are as follows:

Error pattern e	Syndrome vector s
$(0,0,0,0,0,0,0)$	$(0,0,0,0)$
$(0,0,0,0,0,0,1)$	$(\alpha^6,\alpha^5,\alpha^4,\alpha^3)$
$(0,0,0,0,0,0,\alpha)$	$(1,\alpha^6,\alpha^5,\alpha^4)$
$(0,0,0,0,0,0,\alpha^2)$	$(\alpha,1,\alpha^6,\alpha^5)$
...	...
$(0,0,0,0,0,1,0)$	$(\alpha^5,\alpha^3,\alpha,\alpha^6)$
$(0,0,0,0,0,\alpha,0)$	$(\alpha^6,\alpha^4,\alpha^2,1)$
...	...
$(0,0,0,0,0,1,1)$	$(\alpha,\alpha^2,\alpha^2,\alpha^4)$
$(0,0,0,0,0,\alpha,1)$	$(0,1,\alpha,\alpha)$
...	...

Table 4.1: Fragment of syndrome array for $RS(7,3)$ code.

- $r_1(x) = c_1(x) + e(x) = \alpha^3x^6 + \alpha^5x^5 + \alpha^2x^4 + \alpha x^2 + \alpha^3x + \alpha^4$,
- $r_2(x) = c_2(x) + e(x) = \alpha^3x^6 + \alpha^2x^5 + \alpha x^4 + \alpha^4x^3 + \alpha^5x^2 + \alpha^3$,
- $r_3(x) = c_3(x) + e(x) = \alpha^6x^6 + \alpha^6x^4 + \alpha x^2 + \alpha^6x + \alpha^5$.

Syndrome vector $s = (s_1, s_2, s_3, s_4)$ for these received vector is as follows

- $s_1 = r_1(\alpha) = r_2(\alpha) = r_3(\alpha) = 1$,
- $s_2 = r_1(\alpha^2) = r_2(\alpha^2) = r_3(\alpha^2) = \alpha^6$,
- $s_3 = r_1(\alpha^3) = r_2(\alpha^3) = r_3(\alpha^3) = \alpha^5$,
- $s_4 = r_1(\alpha^4) = r_2(\alpha^4) = r_3(\alpha^4) = \alpha^4$.

Values of syndrome vectors for these three received vectors are the same. Syndrome vector for received vector is $s = (1, \alpha^6, \alpha^5, \alpha^4)$. Fragment of syndrome array for $RS(7,3)$ code is shown in table 4.1. For syndrome vector $s = (1, \alpha^6, \alpha^5, \alpha^4)$ there is assigned error pattern $e = (0,0,0,0,0,0,\alpha)$. Received vectors are corrected by adding error pattern to received vector.

4.1.4 Decoding of cyclic code

Generator matrix for cyclic code is as follows [10] [9]:

$$G = \begin{bmatrix} g(x) \\ xg(x) \\ x^2g(x) \\ \vdots \\ x^{k-1}g(x) \end{bmatrix} = \begin{bmatrix} g_0 & g_1 & \dots & g_{2t} & 0 & 0 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & \dots & g_{2t} & 0 & 0 & \dots & 0 \\ 0 & 0 & g_0 & g_1 & \dots & g_{2t} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 & g_0 & g_1 & \dots & g_{2t} \end{bmatrix}$$

Example 4.4.
For $RS(7,3)$ code over $GF(8)$ created by $p(x) = x^3 + x + 1$, there is given following generator polynomial:

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$$

$$G = \begin{bmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{bmatrix}$$

Let information vector be $m = (\alpha, \alpha^2, 1)$. Code for this vector is:

$$\begin{aligned} c = mG &= (\alpha, \alpha^2, 1) \begin{bmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 \\ 0 & 0 & \alpha^3 & \alpha & 1 & \alpha^3 & 1 \end{bmatrix} \\ &= (\alpha^4, \alpha^2, \alpha, \alpha^4, \alpha, 0, 0) + (0, \alpha^5, \alpha^3, \alpha^2, \alpha^5, \alpha^2, 0) + (0, 0, \alpha^3, \alpha, 1, \alpha^3, 1) \\ &= (\alpha^4, \alpha^3, \alpha, 0, \alpha^2, \alpha^5, 1) \end{aligned}$$

Vector $m = (\alpha, \alpha^2, 1)$ can be written as polynomial $m(x) = \alpha + \alpha^2 x + x^2$. Codeword can be created as follows:

$$\begin{aligned} c(x) = m(x)g(x) &= (x^2 + \alpha^2 x + \alpha)(x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3) \\ &= x^6 + \alpha^5 x^5 + \alpha^2 x^4 + \alpha x^2 + \alpha^3 x + \alpha^4 \end{aligned}$$

Results of both methods are the same.

Systematic code can be created, if generator matrix G will be of such form that in the right part there will be identity matrix.

Example 4.5.
To generate generator matrix of systematic code $RS(7,3)$ is used matrix G from example 4.4. This matrix can be transform into such matrix:

$$G' = \begin{bmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ \alpha^6 & \alpha^6 & 1 & \alpha^2 & 0 & 1 & 0 \\ \alpha^5 & \alpha^4 & 1 & \alpha^4 & 0 & 0 & 1 \end{bmatrix}$$

Let information vector be $m = (\alpha, \alpha^2, 1)$. Codeword for this vector is:

$$\begin{aligned} c = mG' &= (\alpha, \alpha^2, 1) \begin{bmatrix} \alpha^3 & \alpha & 1 & \alpha^3 & 1 & 0 & 0 \\ \alpha^6 & \alpha^6 & 1 & \alpha^2 & 0 & 1 & 0 \\ \alpha^5 & \alpha^4 & 1 & \alpha^4 & 0 & 0 & 1 \end{bmatrix} \\ &= (\alpha^4, \alpha^2, \alpha, \alpha^4, \alpha, 0, 0) + (\alpha, \alpha, \alpha^2, \alpha^4, 0, \alpha^2, 0) + (\alpha^5, \alpha^4, 1, \alpha^4, 0, 0, 1) \\ &= (\alpha^3, 0, \alpha^5, \alpha^4, \alpha, \alpha^2, 1) \end{aligned}$$

For the same vector $m \to m(x) = \alpha + \alpha^2 x + x^2$, codeword can be also created as follows:

$$c(x) = m(x)x^{2t} + Q(x) \qquad \text{gdzie } Q(x) = m(x)x^{2t} \bmod g(x)$$

$$\begin{aligned} Q(x) &= m(x)x^4 \bmod g(x) \\ &= (x^6 + \alpha^2 x^5 + \alpha x^4) \bmod (x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3) \\ &= \alpha^4 x^3 + \alpha^5 x^2 + \alpha^3 \end{aligned}$$

$$\begin{aligned} c(x) &= m(x)x^4 + Q(x) = (x^6 + \alpha^2 x^5 + \alpha x^4) + (\alpha^4 x^3 + \alpha^5 x^2 + \alpha^3) \\ &= x^6 + \alpha^2 x^5 + \alpha x^4 + \alpha^4 x^3 + \alpha^5 x^2 + \alpha^3 \end{aligned}$$

Results of both methods are the same.

Generator polynomial of cyclic $RS(n, k)$ code divides polynomial $x^n + 1$. To create parity check matrix H, there must be computed polynomial $h(x)$ of form:

$$h(x) = \frac{x^n + 1}{g(x)}$$

Parity check matrix for $RS(n, k)$ code is as follows:

$$H = \begin{bmatrix} h_k & h_{k-1} & \dots & h_0 & 0 & 0 & \dots & 0 \\ 0 & h_k & h_{k-1} & \dots & h_0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & h_k & h_{k-1} & \dots & h_0 \end{bmatrix} \tag{4.2}$$

It is true that:

$$cH^T = 0$$

Example 4.6.
For $RS(7, 3)$ code, where $g(x) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$:

$$h(x) = \frac{x^7 + 1}{x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3} = x^3 + \alpha^3 x^2 + \alpha^2 x + \alpha^4$$

Parity check matrix for $RS(7, 3)$ code is of form:

$$H = \begin{bmatrix} 1 & \alpha^3 & \alpha^2 & \alpha^4 & 0 & 0 & 0 \\ 0 & 1 & \alpha^3 & \alpha^2 & \alpha^4 & 0 & 0 \\ 0 & 0 & 1 & \alpha^3 & \alpha^2 & \alpha^4 & 0 \\ 0 & 0 & 0 & 1 & \alpha^3 & \alpha^2 & \alpha^4 \end{bmatrix} \tag{4.3}$$

Parity check matrix 4.2 can be of such form, that in the left part is placed identity matrix.

$$H' = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & h'_{k-1} & \dots & h'_0 \\ 0 & 1 & 0 & \dots & 0 & h''_{k-1} & \dots & h''_0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 1 & h'''_{k-1} & \dots & h'''_0 \end{bmatrix} \tag{4.4}$$

Zachodzi:

$$cH'^T = 0$$

Using parity check matrix in this form, there can be generated syndroms, which will be also self error patterns. The constraint is so, that errors must be located on positions $r_0, r_1, \dots, r_{n-k-1}$ of received vector r. Furthermore, there cannot be more errors than error correction capability of code.

Example 4.7.
Parity check matrix 4.3 for $RS(7,3)$ code after some transformations can be written as:

$$H' = \begin{bmatrix} 1 & 0 & 0 & 0 & \alpha^3 & \alpha^6 & \alpha^5 \\ 0 & 1 & 0 & 0 & \alpha & \alpha^6 & \alpha^4 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & \alpha^3 & \alpha^2 & \alpha^4 \end{bmatrix}$$

Given is following received vector:

$$\begin{aligned} r(x) = c(x) + e(x) &= (x^6 + \alpha^5 x^5 + \alpha^2 x^4 + \alpha x^2 + \alpha^3 x + \alpha^4) + (x^3 + \alpha^3 x^2) \\ &= x^6 + \alpha^5 x^5 + \alpha^2 x^4 + x^3 + x^2 + \alpha^3 x + \alpha^4 \end{aligned}$$

Using parity check matrix, there can be computed syndrome vector:

$$\begin{aligned} s' = rH'^T = (\alpha^4, \alpha^3, 1, 1, \alpha^2, \alpha^5, 1) &\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \alpha^3 & \alpha & 1 & \alpha^3 \\ \alpha^6 & \alpha^6 & 1 & \alpha^2 \\ \alpha^5 & \alpha^4 & 1 & \alpha^4 \end{bmatrix} \\ = (\alpha^4, 0, 0, 0) + (0, \alpha^3, 0, 0) + (0, 0, 1, 0) + (0, 0, 0, 1) \\ +(\alpha^5, \alpha^3, \alpha^2, \alpha^5) + (\alpha^4, \alpha^4, \alpha^5, 1) + (\alpha^5, \alpha^4, 1, \alpha^4) = (0, 0, \alpha^3, 1) \end{aligned}$$

Syndrome in polynomial form $s' = (0, 0, \alpha^3, 1) \to s'(x) = x^3 + \alpha^3 x^2$ is the same as error vector $e(x)$. Received vector can be corrected as follows:

$$\begin{aligned} c(x) &= r(x) - e(x) = r(x) - s'(x) \\ &= (x^6 + \alpha^5 x^5 + \alpha^2 x^4 + x^3 + x^2 + \alpha^3 x + \alpha^4) - (x^3 + \alpha^3 x^2) \\ &= x^6 + \alpha^5 x^5 + \alpha^2 x^4 + \alpha x^2 + \alpha^3 x + \alpha^4 \end{aligned}$$

For cyclic codes it is true that:

$$m(x)g(x) + s'(x) = r(x)$$

$$s'(x) = r(x) \bmod g(x)$$

It is also true that:

$$r(x) = c(x) + e(x) \qquad \text{and} \qquad r(x) = c(x) + s'(x)$$

$$e(x) = r(x) \bmod g(x)$$

From above equations it can be noticed that $e(x) = s'(x)$. It is true when errors are on positions $r_0, r_1, \ldots, r_{n-k-1}$ of received vector r.

Decoding algorithm for cyclic codes:

1. Let $i = 0$ and i denotes number of shift of received vector in left direction.
2. Let $r_i(x)$ denotes received vector $r(x)$ and shifted i-times in left direction. Compute:
$$s'_i(x) = r_i(x) \bmod g(x)$$
3. Let $w(y)$ denotes value of Hamming weight of vector y. Hamming weight describres number of nonzero elements of vector. Compute:
$$w = w(s'_i(x))$$
4. If $w \leq t$, where t denotes error correction capability of code, then compute:
$$r_i(x) = r_i(x) - s'_i(x) \ ,$$
then shift $r_i(x)$ in right direction i-times and finish. If $w > t$, then continue.
5. If $i = k$, where k denotes number of control elements of codeword, then finish and signal that not correctable error pattern occured. If $i < k$, then continue.
6. Shift cyclically one time $r_i(x)$ in left direction.
7. Increase i by 1:
$$i = i + 1$$
8. Return to point 2.

4.2 Finding position of errors

Reed-Solomon can be treated as nonbinary BCH code, which means that code symbols are nonbinary. During decoding binary codes there is needed one information to correct error, which is error position. These errors can be corrected by adding 1 to value on error position. For nonbinary codes this is a bit more complicated, because except error position, there must be known error value also. In this chapter, there are described algorithms for finding error positions and error values.

4.2.1 Peterson-Gorenstein-Zierler algorithm

Peterson-Gorenstein-Zierler algorithm finds error positions [10] [2] [3]. Using definition 4.1, it is true that:

$$s_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i)$$

For consecutive $2t$ roots of code generator polynomial, it is true that:

$$s_i = c(\alpha^i) = 0$$

Syndrome can be expressed as:

$$s_i = e(\alpha^i) = \sum_{j=0}^{n-1} e_j (\alpha^i)^j$$

Error occurs in place, where $e_j \neq 0$, so for number of occured errors v, error positions are denoted as $k_1, k_2, \ldots, k_v$:

$$s_i = \sum_{j=1}^{v} e_{k_j} (\alpha^i)^{k_j} = \sum_{j=1}^{v} e_{k_j} (\alpha^{k_j})^i$$

Let $X_j = \alpha^{k_j}$. This value is dependent from error positions and is called error locator:

$$s_i = \sum_{j=1}^{v} e_{k_j} (X_j)^i \tag{4.5}$$

Consecutive $2t$ values of syndromes can be expressed as:

$$\begin{aligned}
s_1 &= e_{k_1} X_1 + e_{k_2} X_2 + \ldots + e_{k_v} X_v \\
s_2 &= e_{k_1} X_1^2 + e_{k_2} X_2^2 + \ldots + e_{k_v} X_v^2 \\
&\ldots \\
s_{2t} &= e_{k_1} X_1^{2t} + e_{k_2} X_2^{2t} + \ldots + e_{k_v} X_v^{2t}
\end{aligned}$$

Let there be a polynomial with roots of multiplicative inverses of X_j. Such polynomial is called error-locator polynomial:

$$\Lambda(x) = \prod_{j=1}^{v}(1 - X_j x) = \Lambda_v x^v + \Lambda_{v-1}x^{v-1} + \ldots + \Lambda_1 x + \Lambda_0 \tag{4.6}$$

Looking at definition of error-locator polynomial 4.6, it can be noticed that $\Lambda_0 = 1$. kth syndrome can be expressed as [12]:

$$\Lambda_v s_{k-v} + \Lambda_{v-1}s_{k-v+1} + \ldots + \Lambda_1 s_{k-1} = -s_k \tag{4.7}$$

Example 4.8.
For $RS(7,3)$ code of error-correction capability $t = 2$, let number of occured errors be $v = t$. It is true that $\Lambda_0 = 1$ and polynomial $\Lambda(x)$ should be of degree 2:

$$\Lambda(x) = \Lambda_2 x^2 + \Lambda_1 x + 1$$

Using equation 4.7, it can be computed:

$$\begin{aligned} \text{dla } k = 3 \qquad \Lambda_2 s_{3-2} + \Lambda_{2-1}s_{3-2+1} &= -s_3 \\ \Lambda_2 s_1 + \Lambda_1 s_2 &= -s_3 \end{aligned}$$

$$\begin{aligned} \text{dla } k = 4 \qquad \Lambda_2 s_{4-2} + \Lambda_{2-1}s_{4-2+1} &= -s_4 \\ \Lambda_2 s_2 + \Lambda_1 s_3 &= -s_4 \end{aligned}$$

Assuming that number of occured errors is v, next v equations 4.7 can be expressed as:

$$M^{(v)}\Lambda^{(v)} = \begin{bmatrix} s_1 & s_2 & \ldots & s_v \\ s_2 & s_3 & \ldots & s_{v+1} \\ \vdots & \vdots & \vdots & \vdots \\ s_v & s_{v+1} & \ldots & s_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -s_{v+1} \\ -s_{v+2} \\ \vdots \\ -s_{2v} \end{bmatrix} \tag{4.8}$$

If there are known $2t$ consecutive syndrome values $s_1, s_2, \ldots, s_{2t-1}, s_{2t}$ and it is not true that $s_1 = s_2 = \ldots = s_{2t} = 0$, then Peterson-Gorenstein-Zierler algorithm is as follows:

1. Assume that $v = t$.
2. Check if matrix $M^{(v)}$ is singular.

- If determinant of matrix $M^{(v)}$ is equal to 0, which means that $\det M^{(v)} = 0$, then matrix is singular. In this case decrease value v by 1, $v = v - 1$. If $v = 0$, then finish and signal that not correctable error pattern has occured, else return to point 2.
- If determinant of matrix $M^{(v)}$ is not equal to 0, which means that $\det M^{(v)} \neq 0$, then matrix is nonsingular and continue.

3. Solve following equation set:

$$\begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -s_{v+1} \\ -s_{v+2} \\ \vdots \\ -s_{2v} \end{bmatrix} \begin{bmatrix} s_1 & s_2 & \dots & s_v \\ s_2 & s_3 & \dots & s_{v+1} \\ \vdots & \vdots & \vdots & \vdots \\ s_v & s_{v+1} & \dots & s_{2v-1} \end{bmatrix}^{-1}$$

4. The result is error-location polynomial which is:

$$\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1} x^{v-1} + \ldots + \Lambda_1 x + 1$$

Determinant of matrix $M^{(v)}$ can be computed by Laplace expansion. Let there be a matrix of m rows and columns:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}$$

Laplace expansion can be expressed as follows:

$$\det A = \det \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & \dots & a_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,m} \end{bmatrix} = \sum_{j=1}^{m} a_{i,j} D_{i,j}$$

Index $1 \leq i \leq m$ denotes number of row of matrix A. Cofactor $D_{i,j}$ of element $a_{i,j}$ can be expressed as:

$$D_{i,j} = (-1)^{i+j} \det A_{i,j},$$

where $A_{i,j}$ denotes matrix A od order $m-1$ which is result of erasing ith row and jth column of matrix A [11]. In Galois field of characteristic 2, it can be reduced that $(-1)^{i+j} = 1$ for each row i and column j, so cofactor can be expressed as:

$$D_{i,j} = \det A_{i,j}$$

Equations sets can be computed by Gaussian elimination. Let there be a following set of equations:

$$AX = B$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,m} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,m} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,m} \end{bmatrix} \begin{bmatrix} x_m \\ x_{m-1} \\ \vdots \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Algorithm of Gaussian elimination is as follows:

1. Move matrix B to the right side of matrix A:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,m} & b_m \\ a_{2,1} & a_{2,2} & \ldots & a_{2,m} & b_{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m,1} & a_{m,2} & \ldots & a_{m,m} & b_1 \end{bmatrix}$$

2. Set variable $i = 1$.
3. Multiply ith row by $a_{i,i}^{-1}$.
4. For $1 \leq j \leq m; i \neq j$ compute $r_j = a_{j,i}r_i + r_j$, where r_j denotes jth row.
5. Increase i by 1, $i = i + 1$. If $i > m$ then finish, else go to point 3.

The result is following matrix:

$$\begin{bmatrix} 1 & 0 & \ldots & 0 & x_m \\ 0 & 1 & \ldots & 0 & x_{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & x_1 \end{bmatrix}$$

Example 4.9.
For $RS(15, 7)$ code let exemplary codeword be:

$$c(x) = \alpha^6 x^{14} + \alpha x^{13} + x^{12} + \alpha^2 x^{11} + \alpha x^{10} + \alpha x^9 \\ + \alpha^2 x^8 + \alpha^3 x^7 + \alpha^7 x^6 + \alpha^9 x^5 + \alpha^5 x^4 + \alpha^9 x^3 + x + \alpha^{10}$$

Error vector is as follows:

$$e(x) = \alpha x^{14} + \alpha^2 x^9 + x^4$$

Received vector is as follows:

$$r(x) = \alpha^{11} x^{14} + \alpha x^{13} + x^{12} + \alpha^2 x^{11} + \alpha x^{10} + \alpha^5 x^9 \\ + \alpha^2 x^8 + \alpha^3 x^7 + \alpha^7 x^6 + \alpha^9 x^5 + \alpha^{10} x^4 + \alpha^9 x^3 + x + \alpha^{10}$$

$RS(15,7)$ code has error correction capability of $t=4$. First step of finding error position is to compute $2t$ syndrome values. Roots of code generator polynomial are consecutive elements $\alpha, \alpha^2, \ldots, \alpha^{2t}$.
$s_1 = r(\alpha) = \alpha^6$
$s_2 = r(\alpha^2) = \alpha^9$
$s_3 = r(\alpha^3) = \alpha^7$
$s_4 = r(\alpha^4) = \alpha^3$
$s_5 = r(\alpha^5) = \alpha^6$
$s_6 = r(\alpha^6) = \alpha^4$
$s_7 = r(\alpha^7) = 1$
$s_8 = r(\alpha^8) = \alpha^3$

Next stage is to check if matrix $M^{(v=t=4)}$ is nonsingular:

$$M^{(4)} = \begin{bmatrix} \alpha^6 & \alpha^9 & \alpha^7 & \alpha^3 \\ \alpha^9 & \alpha^7 & \alpha^3 & \alpha^6 \\ \alpha^7 & \alpha^3 & \alpha^6 & \alpha^4 \\ \alpha^3 & \alpha^6 & \alpha^4 & 1 \end{bmatrix}$$

$$
\begin{aligned}
\det M^{(4)} &= \det \begin{bmatrix} \alpha^6 & \alpha^9 & \alpha^7 & \alpha^3 \\ \alpha^9 & \alpha^7 & \alpha^3 & \alpha^6 \\ \alpha^7 & \alpha^3 & \alpha^6 & \alpha^4 \\ \alpha^3 & \alpha^6 & \alpha^4 & 1 \end{bmatrix} \\
&= \alpha^6 \det \begin{bmatrix} \alpha^7 & \alpha^3 & \alpha^6 \\ \alpha^3 & \alpha^6 & \alpha^4 \\ \alpha^6 & \alpha^4 & 1 \end{bmatrix} + \alpha^9 \det \begin{bmatrix} \alpha^9 & \alpha^3 & \alpha^6 \\ \alpha^7 & \alpha^6 & \alpha^4 \\ \alpha^3 & \alpha^4 & 1 \end{bmatrix} \\
&+ \alpha^7 \det \begin{bmatrix} \alpha^9 & \alpha^7 & \alpha^6 \\ \alpha^7 & \alpha^3 & \alpha^4 \\ \alpha^3 & \alpha^6 & 1 \end{bmatrix} + \alpha^3 \det \begin{bmatrix} \alpha^9 & \alpha^7 & \alpha^3 \\ \alpha^7 & \alpha^3 & \alpha^6 \\ \alpha^3 & \alpha^6 & \alpha^4 \end{bmatrix} \\
&= \alpha^6(\alpha^7 \det \begin{bmatrix} \alpha^6 & \alpha^4 \\ \alpha^4 & 1 \end{bmatrix} + \alpha^3 \det \begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & 1 \end{bmatrix} + \alpha^6 \det \begin{bmatrix} \alpha^3 & \alpha^6 \\ \alpha^6 & \alpha^4 \end{bmatrix}) \\
&+ \alpha^9(\alpha^9 \det \begin{bmatrix} \alpha^6 & \alpha^4 \\ \alpha^4 & 1 \end{bmatrix} + \alpha^3 \det \begin{bmatrix} \alpha^7 & \alpha^4 \\ \alpha^3 & 1 \end{bmatrix} + \alpha^6 \det \begin{bmatrix} \alpha^7 & \alpha^6 \\ \alpha^3 & \alpha^4 \end{bmatrix}) \\
&+ \alpha^7(\alpha^9 \det \begin{bmatrix} \alpha^3 & \alpha^4 \\ \alpha^6 & 1 \end{bmatrix} + \alpha^7 \det \begin{bmatrix} \alpha^7 & \alpha^4 \\ \alpha^3 & 1 \end{bmatrix} + \alpha^6 \det \begin{bmatrix} \alpha^7 & \alpha^3 \\ \alpha^3 & \alpha^6 \end{bmatrix}) \\
&+ \alpha^3(\alpha^9 \det \begin{bmatrix} \alpha^3 & \alpha^6 \\ \alpha^6 & \alpha^4 \end{bmatrix} + \alpha^7 \det \begin{bmatrix} \alpha^7 & \alpha^6 \\ \alpha^3 & \alpha^4 \end{bmatrix} + \alpha^3 \det \begin{bmatrix} \alpha^7 & \alpha^3 \\ \alpha^3 & \alpha^6 \end{bmatrix}) \\
&= \alpha^6(\alpha^7(\alpha^6 + \alpha^8) + \alpha^3(\alpha^3 + \alpha^{10}) + \alpha^6(\alpha^7 + \alpha^{12})) \\
&+ \alpha^9(\alpha^9(\alpha^6 + \alpha^8) + \alpha^3(\alpha^7 + \alpha^7) + \alpha^6(\alpha^{11} + \alpha^9)) \\
&+ \alpha^7(\alpha^9(\alpha^3 + \alpha^{10}) + \alpha^7(\alpha^7 + \alpha^7) + \alpha^6(\alpha^{13} + \alpha^6)) \\
&+ \alpha^3(\alpha^9(\alpha^7 + \alpha^{12}) + \alpha^7(\alpha^{11} + \alpha^9) + \alpha^3(\alpha^{13} + \alpha^6)) \\
&= 0
\end{aligned}
$$

Determinant of matrix $M^{(4)}$ is equal to zero, which means that matrix is singular. Next step is to compute determinant of matrix $M^{(3)}$:

$$\det M^{(3)} = \det \begin{bmatrix} \alpha^6 & \alpha^9 & \alpha^7 \\ \alpha^9 & \alpha^7 & \alpha^3 \\ \alpha^7 & \alpha^3 & \alpha^6 \end{bmatrix} = \alpha^9$$

Determinant of matrix $M^{(3)}$ is not equal to zero, which means that matrix is nonsingular. It means also that three errors has occured. There must be following set of equations:

$$\begin{bmatrix} \alpha^6 & \alpha^9 & \alpha^7 \\ \alpha^9 & \alpha^7 & \alpha^3 \\ \alpha^7 & \alpha^3 & \alpha^6 \end{bmatrix} \begin{bmatrix} \Lambda_3 \\ \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} \alpha^3 \\ \alpha^6 \\ \alpha^4 \end{bmatrix}$$

Using Gaussian elimination there are computed coefficients of polynomial $\Lambda(x)$:

$$\begin{bmatrix} \alpha^6 & \alpha^9 & \alpha^7 & \alpha^3 \\ \alpha^9 & \alpha^7 & \alpha^3 & \alpha^6 \\ \alpha^7 & \alpha^3 & \alpha^6 & \alpha^4 \end{bmatrix}$$

Variable $i = 1$, so first row is multiplied by $a_{i,i}^{-1} = a_{1,1}^{-1} = \alpha^{-6} = \alpha^9$:

$$\begin{bmatrix} 1 & \alpha^3 & \alpha & \alpha^{12} \\ \alpha^9 & \alpha^7 & \alpha^3 & \alpha^6 \\ \alpha^7 & \alpha^3 & \alpha^6 & \alpha^4 \end{bmatrix}$$

To the second row is added first row multiplied by α^9, and to the third row is added first row multiplied by α^7.

$$\begin{bmatrix} 1 & \alpha^3 & \alpha & \alpha^{12} \\ 0 & \alpha^2 & \alpha^{12} & 0 \\ 0 & \alpha^{12} & \alpha^{14} & 0 \end{bmatrix}$$

As the result of next steps following matrix is obtained:

$$\begin{bmatrix} 1 & 0 & 0 & \alpha^{12} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Result of set of equations is $\Lambda_3 = \alpha^{12}$, $\Lambda_2 = 0$, $\Lambda_1 = 0$. The error locator polynomial is as follows:

$$\Lambda(x) = \alpha^{12}x^3 + 1$$

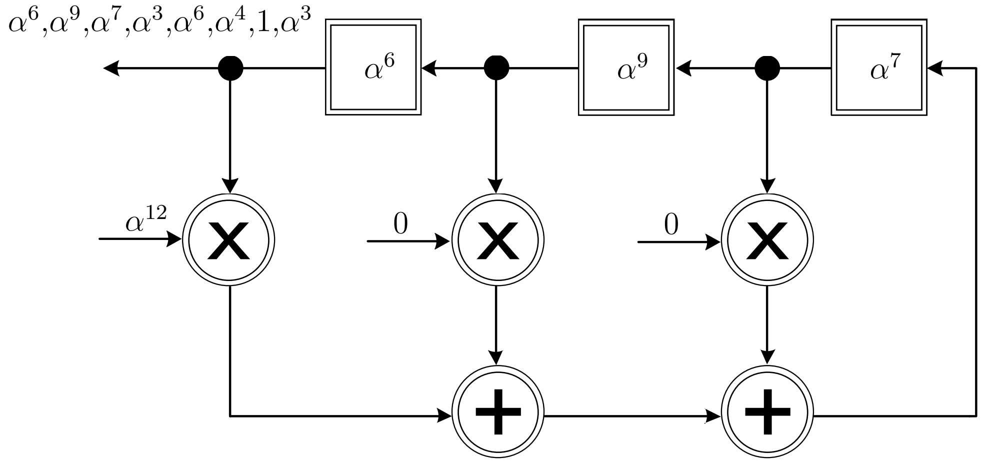
Roots of this polynomial are $x_1 = \alpha$, $x_2 = \alpha^6$, $x_3 = \alpha^{11}$. Error locators are $X_1 = x_1^{-1} = \alpha^{14}$, $X_2 = x_2^{-1} = \alpha^9$, $X_3 = x_3^{-1} = \alpha^4$. Error locator polynomial can be expressed as:

$$\Lambda(x) = (\alpha^{14}x - 1)(\alpha^9 x - 1)(\alpha^4 x - 1)$$

It means that errors have occured on positions 14, 9 and 4 of received vector.

4.2.2 Berlekamp-Massey algorithm

Berlekamp-Massey algorithm finds error-locator polynomial [10] [14] [4] [5]. Using equation 4.7 for some first elements of code of error-correction capa-



α^6,α^9,α^7,α^3,α^6,α^4,1,α^3
α^6
α^9
α^7
α^{12}
0
0

Figure 4.1: LFSR generates next syndromes with use of polynomial $\Lambda(x) = \alpha^{12}x^3 + 1$.

bility of t, it can be expressed for v and $v < j \leq 2t$ as:

$$\begin{aligned}
-s_{v+1} &= \Lambda_1 s_v + \Lambda_2 s_{v-1} + \ldots + \Lambda_v s_1 \\
-s_{v+2} &= \Lambda_1 s_{v+1} + \Lambda_2 s_v + \ldots + \Lambda_v s_2 \\
&\ldots \\
-s_{2t} &= \Lambda_1 s_{2t-1} + \Lambda_2 s_{2t-2} + \ldots + \Lambda_v s_{2t-v}
\end{aligned}$$

Example 4.10.
Give is $RS(15, 7)$ code of error-correction capability $t = 4$. In received vector occured 3 errors. There was computed following error-locatory polynomial:

$$\Lambda(x) = \alpha^{12}x^3 + 1$$

Next $2t$ syndromes for received vector:
$s_1 = \alpha^6$
$s_2 = \alpha^9$
$s_3 = \alpha^7$
$s_4 = \alpha^3$
$s_5 = \alpha^6$
$s_6 = \alpha^4$
$s_7 = 1$
$s_8 = \alpha^3$

For number of errors $v = \deg \Lambda(x) = 3$, it is true that:

$$\begin{aligned} -s_4 &= \Lambda_1 s_3 + \Lambda_2 s_2 + \Lambda_3 s_1 \\ \alpha^3 &= 0 \cdot \alpha^7 + 0 \cdot \alpha^9 + \alpha^{12} \cdot \alpha^6 \end{aligned}$$

$$\begin{aligned} -s_5 &= \Lambda_1 s_4 + \Lambda_2 s_3 + \Lambda_3 s_2 \\ \alpha^6 &= 0 \cdot \alpha^3 + 0 \cdot \alpha^7 + \alpha^{12} \cdot \alpha^9 \end{aligned}$$

$$\begin{aligned} -s_6 &= \Lambda_1 s_5 + \Lambda_2 s_4 + \Lambda_3 s_3 \\ \alpha^4 &= 0 \cdot \alpha^6 + 0 \cdot \alpha^3 + \alpha^{12} \cdot \alpha^7 \end{aligned}$$

$$\begin{aligned} -s_7 &= \Lambda_1 s_6 + \Lambda_2 s_5 + \Lambda_3 s_4 \\ 1 &= 0 \cdot \alpha^4 + 0 \cdot \alpha^6 + \alpha^{12} \cdot \alpha^3 \end{aligned}$$

$$\begin{aligned} -s_8 &= \Lambda_1 s_7 + \Lambda_2 s_6 + \Lambda_3 s_5 \\ \alpha^3 &= 0 \cdot 1 + 0 \cdot \alpha^4 + \alpha^{12} \cdot \alpha^6 \end{aligned}$$

LFSR (Linear Feedback Shift Register) can be used to compute next syndromes. LFSR, which generates next syndromes $s_1, s_2, \ldots, s_8$ is shown on picture 4.1. At the beginning flip-flops are initialized with syndromes s_1, s_2, s_3. Multiplicators multiply by constant, which is one of coefficient of error-locator polynomial $\Lambda(x)$. With each clock cycle there is generated next value of syndrome.

Problem of finding coefficients of error-locator polynomial can be expressed as problem of construction such LFSR, which generates next syndromes $s_1, s_2, \ldots, s_{2t}$. Berlekamp-Massey algorithm generates such LFSR during $2t$ iterations.

Berlekamp-Massey algorithm is as follows for $s_1, s_2, \ldots, s_{2t}$, where it is not true that $s_1 = s_2 = \ldots = s_{2t} = 0$:

1. Set variable $L_0 = 0$. Variable L_0 represents length of LFSR in rth iteration. Set variable $r = 0$. Variable r represents iteration counter. Set error-locator polynomial as $\Lambda^{(0)}(x) = 1$. Set auxiliary polynomial as $B(x) = 1$, which will be used to construct LFSR.
2. Increase iteration counter r by 1, $r = r + 1$.
3. Compute value of discrepancy $\Delta^{(r)}$. Discrepancy is defined as difference between value of syndrome s_r, which should be produced by

LFSR and value, which is really produced by LFSR:

$$\Delta^{(r)} = s_r - \sum_{j=1}^{L_{r-1}} \Lambda_j^{(r-1)} s_{r-j}$$

Recall that $\Lambda_0 = 1$:

$$\Delta^{(r)} = \sum_{j=0}^{L_{r-1}} \Lambda_j^{(r-1)} s_{r-j}$$

4. Compare $\Delta^{(r)}$ with zero:
   - If $\Delta^{(r)} = 0$, then it means that LFSR really produces syndrome s_r and it shouldn't be modified, so $\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x)$, $L_r = L_{r-1}$.
   - If $\Delta^{(r)} \neq 0$, then it means, that LFSR should be modified. There can be modified two things – length of LFSR and coefficients of polynomial $\Lambda^{(r-1)}(x)$. There are computed new coefficients of $\Lambda^{(r)}(x)$:
   $$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - \Delta^{(r)} x B(x)$$
   New length of LFSR in given iteration r is described as:
   $$L_r = \max\left[L_{r-1}, r - L_{r-1}\right]$$
   It is true, that length of LFSR must be changed, if:
   $$2L_{r-1} \leq r - 1$$
     – if $2L_{r-1} \leq r - 1$, then:
     $$L_r = r - L_{r-1}$$
     $$B(x) = \frac{\Lambda^{(r-1)}(x)}{\Delta^{(r)}}$$
     – if $2L_{r-1} > r - 1$, then:
     $$L_r = L_{r-1}$$
     $$B(x) = xB(x)$$
5. If $\deg \Lambda^{(r)}(x) > t$, where t denotes error correction capability of code, then finish and signal that not correctable error pattern has occured, else continue.
6. If $r = 2t$, then finish. The error-locator polynomial is $\Lambda(x) = \Lambda^{(2t)}(x)$. If $r < 2t$, then return to point 2.

Example 4.11.
Give are following syndromes:
$s_1 = \alpha^6$
$s_2 = \alpha^9$
$s_3 = \alpha^7$
$s_4 = \alpha^3$
$s_5 = \alpha^6$
$s_6 = \alpha^4$
$s_7 = 1$
$s_8 = \alpha^3$
With use of Berlekamp-Massey algorithm, there will be created LFSR, which generates these syndromes, and error-locator polynomial $\Lambda(x)$.
First, variables are initialized:

$$\Lambda^{(0)}(x) = 1 \qquad B(x) = 1 \qquad L_0 = 0 \qquad r = 0$$

Variable r is increased by 1, $r = 1$
Discrepancy is computed:

$$\Delta^{(1)} = \sum_{j=0}^{L_{r-1}} \Lambda_j^{(r-1)} s_{r-j} = \sum_{j=0}^{0} \Lambda_j^{(0)} s_{1-j} = \Lambda_0^{(0)} s_1 = \alpha^6$$

Discrepancy is not equal to zero, then LFSR should be modified:

$$\Lambda^{(1)}(x) = \Lambda^{(r-1)}(x) - \Delta^{(r)} B(x) = \Lambda^{(0)}(x) - \Delta^{(1)} B(x) x = 1 - \alpha^6 \cdot 1x = \alpha^6 x + 1$$

It is satisfied also that:

$$2L_{r-1} \leq r - 1$$
$$0 \leq 0$$

Length of LFSR should be also changed and auxiliary polynomial $B(x)$ is modified:

$$L_r = r - L_{r-1}$$
$$L_1 = 1 - 0 = 1$$
$$B(x) = \frac{\Lambda^{(r-1)}(x)}{\Delta^{(r)}} = \frac{\Lambda^{(0)}(x)}{\Delta^{(1)}}$$
$$B(x) = 1 \cdot \alpha^{-6} = \alpha^9$$

Iteration counter is increased sequentially and algorithms finishes works when $r = 8$. Result of algorithm is $\Lambda(x) = \Lambda^{(8)}(x) = \alpha^{12}x^3 + 1$. Changes of variables in each iteration are show in table 4.2. Result of algorithm is the same as for example 4.9, where for identical input data computed was error-locator polynomial with use of Peterson-Gorensteina-Zierler algorithm.

r	L_r	s_r	$\Delta^{(r)}$	$\Lambda^{(r)}(x)$	$B(x)$
0	0	–	–	1	1
1	1	α^6	α^6	$\alpha^6 x + 1$	α^9
2	1	α^9	α^8	$\alpha^3 x + 1$	$\alpha^9 x$
3	2	α^7	α^2	$\alpha^{11}x^2 + \alpha^3 x + 1$	$\alpha x + \alpha^{13}$
4	2	α^3	α^{14}	$\alpha^{12}x^2 + \alpha^{10}x + 1$	$\alpha x^2 + \alpha^{13}x$
5	3	α^6	α	$\alpha^2x^3 + \alpha^5x^2 + \alpha^{10}x + 1$	$\alpha^{11}x^2 + \alpha^9 x + \alpha^{14}$
6	3	α^4	α^{11}	$\alpha^{12}x^3 + 1$	$\alpha^{11}x^3 + \alpha^9x^2 + \alpha^{14}x$
7	3	1	0	$\alpha^{12}x^3 + 1$	$\alpha^{11}x^4 + \alpha^9x^3 + \alpha^{14}x^2$
8	3	α^3	0	$\alpha^{12}x^3 + 1$	$\alpha^{11}x^5 + \alpha^9x^4 + \alpha^{14}x^3$

Table 4.2: Next steps for Berlekamp-Massey algorithm for example 4.11

4.2.3 Sugiyama algorithm

Sugiyama algorithm is used to find error-locator polynomial and it employs extended Euclidean algorithm to solve this task [14]. Given is following polynomial, where coefficients are next syndromes:

$$S(x) = s_1x + s_2x^2 + s_3x^3 + \ldots + s_{2t}x^{2t} = \sum_{n=1}^{2t} s_n x^n$$

Given is following equation:

$$\Lambda(x)S(x) = \Omega(x) \bmod x^{2t+1}$$

This equation is called key equation. Polynomial $\Omega(x)$ is called error-evaluator polynomial and is ussed to find error values. Above equation can be expressed as:

$$a(x)x^{2t+1} + \Lambda(x)S(x) = \Omega(x),$$

where $a(x)$ is some polynomial which satisfy equation.
Bezout identity for two polynomials $a(x)$ and $b(x)$, for which greatest common divisor is $GCD(a(x), b(x)) = c(x)$, can be expressed as:

$$a(x)m(x) + b(x)n(x) = c(x),$$

where $m(x)$ and $n(x)$ are some polynomials. This equation can be solved by extended Euclidean algorithm. Result of algorithm is error-locator polynomial $\Lambda(x)$.
Sugiyama algorithm for syndrome polynomial $S(x)$, where $\deg S(x) > 0$, is as follows:

1. Set variables $s(x) = x^{2t+1}$, $t(x) = S(x)$, $A(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
2. Check if degree of $t(x)$ is less or equal than error correction capability of code t:
   - If $\deg t(x) \le t$, finish. The result is:
   $$\Lambda(x) = (A_{2,2}(0))^{-1} A_{2,2}(x)$$
   If $\deg \Lambda(x) = 0$, signal that not correctable error pattern has occured.
   - Jeśli $\deg t(x) > t$, then continue.
3. Compute:
$$Q(x) = \frac{s(x)}{t(x)}$$
4. Compute:
$$\begin{bmatrix} s(x) \\ t(x) \end{bmatrix} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix}$$
5. Oblicz:
$$A(x) \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} A(x)$$
6. Return to point 2.

Example 4.12.
Given are following syndromes:
$s_1 = \alpha^6$
$s_2 = \alpha^9$
$s_3 = \alpha^7$
$s_4 = \alpha^3$
$s_5 = \alpha^6$
$s_6 = \alpha^4$
$s_7 = 1$
$s_8 = \alpha^3$
Initialization of variables:
$s(x) = x^9$
$t(x) = S(x) = \alpha^3x^8 + x^7 + \alpha^4x^6 + \alpha^6x^5 + \alpha^3x^4 + \alpha^7x^3 + \alpha^9x^2 + \alpha^6x$
Algorithms proceeds for $RS(15, 7)$ code of error-correction capability $t = 4$.
$$\deg t(x) = 8 > t = 4,$$
so algorithm continues. Compute:
$$\begin{aligned} Q(x) = \frac{s(x)}{t(x)} &= \frac{x^9}{\alpha^3x^8 + x^7 + \alpha^4x^6 + \alpha^6x^5 + \alpha^3x^4 + \alpha^7x^3 + \alpha^9x^2 + \alpha^6x} \\ &= \alpha^{12}x + \alpha^9 \end{aligned}$$

Compute:

$$\begin{bmatrix} s(x) \\ t(x) \end{bmatrix} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & \alpha^{12}x+\alpha^9 \end{bmatrix} \begin{bmatrix} x^9 \\ \alpha^3x^8+x^7+\alpha^4x^6+\alpha^6x^5+\alpha^3x^4+\alpha^7x^3+\alpha^9x^2+\alpha^6x \end{bmatrix}$$

$$= \begin{bmatrix} \alpha^3x^8+x^7+\alpha^4x^6+\alpha^6x^5+\alpha^3x^4+\alpha^7x^3+\alpha^9x^2+\alpha^6x \\ \alpha^3x^7+\alpha^8x^6+\alpha^6x^4+\alpha^{11}x^3+x \end{bmatrix}$$

Compute:

$$A(x) \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} A(x) = \begin{bmatrix} 0 & 1 \\ 1 & \alpha^{12}x+\alpha^9 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ 1 & \alpha^{12}x+\alpha^9 \end{bmatrix}$$

Algorithm continues until $\deg t(x) \leq t$. The result of algorithm is $\Lambda(x) = \alpha^{12}x^3+1$, which is the same as result of Peterson-Gorenstein-Zierler algorithm and Berlekamp-Massey algorithm for the same input data.

4.2.4 Chien search

Chien search is a method of finding roots of error-locator polynomial $\Lambda(x)$ [13] [14]. For field $GF(2^m)$ there are check next elements $1, \alpha, \ldots, \alpha^{2^m-2}$ whether they are roots or not of $\Lambda(x)$.

Polynomial $\Lambda(x)$ can be expressed as:

$$\Lambda(x) = \Lambda_t x^t + \Lambda_{t-1}x^{t-1} + \ldots + \Lambda_1 x + \Lambda_0$$

Element α^i is root of polynomial $\Lambda(x)$ if it satisfies:

$$\Lambda(\alpha^i) = \Lambda_t \alpha^{it} + \Lambda_{t-1}\alpha^{i(t-1)} + \ldots + \Lambda_1\alpha^i + \Lambda_0 = 0$$

Recall that $\Lambda_0 = 1$:

$$\Lambda(\alpha^i) - 1 = \Lambda_t \alpha^{it} + \Lambda_{t-1}\alpha^{i(t-1)} + \ldots + \Lambda_1\alpha^i = -1$$

It is true for element α^{i+1} that:

$$\Lambda(\alpha^{i+1}) - 1 = \Lambda_t \alpha^{(i+1)t} + \Lambda_{t-1}\alpha^{(i+1)(t-1)} + \ldots + \Lambda_1\alpha^{i+1}$$

$$\Lambda(\alpha^{i+1}) - 1 = \Lambda_t \alpha^{it}\alpha^t + \Lambda_{t-1}\alpha^{i(t-1)}\alpha^{t-1} + \ldots + \Lambda_1\alpha^i\alpha$$

It can be noticed that value of $\Lambda(\alpha^{i+1})$ can be computed with use of previously computed value of $\Lambda(\alpha^i)$. If value of polynomial $(\Lambda(\alpha^i) - 1)$ is equal to -1, then it means, that element α^i is root of polynomial $\Lambda(x)$.

4.3 Finding error values

Using definition of syndrome 4.5, it is true that:

$$s_i = \sum_{j=1}^{v} e_{k_j} (X_j)^i, \tag{4.9}$$

Next $2t$ values of syndromes can be expressed as:

$$\begin{aligned}
s_1 &= e_{k_1} X_1 + e_{k_2} X_2 + \ldots + e_{k_v} X_v \\
s_2 &= e_{k_1} X_1^2 + e_{k_2} X_2^2 + \ldots + e_{k_v} X_v^2 \\
&\ldots \\
s_{2t} &= e_{k_1} X_1^{2t} + e_{k_2} X_2^{2t} + \ldots + e_{k_v} X_v^{2t}
\end{aligned}$$

After finding of error-locator polynomial $\Lambda(x)$, there are known number of occured errors which is $v = \deg \Lambda(x)$, error locators $X_1, X_2, \ldots, X_v$ and next syndromes $s_1, s_2, \ldots, s_{2t}$. Above equation limited to v syndromes can be expressed as [12]:

$$\begin{bmatrix} X_1 & X_2 & \ldots & X_v \\ X_1^2 & X_2^2 & \ldots & X_v^2 \\ \vdots & \vdots & \vdots & \vdots \\ X_1^v & X_2^v & \ldots & X_v^v \end{bmatrix} \begin{bmatrix} e_{k_1} \\ e_{k_2} \\ \vdots \\ e_{k_v} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_v \end{bmatrix}$$

Above set of equations can be solved with use of Gaussian elimination. Recall that error locator is of form $X_j = \alpha^{k_j}$, then error vector can be expressed as:

$$e(x) = e_{k_1} x^{\log_\alpha X_1} + e_{k_2} x^{\log_\alpha X_2} + \ldots + e_{k_v} x^{\log_\alpha X_v}$$

Example 4.13.
Given are following syndromes:
$s_1 = \alpha^6$
$s_2 = \alpha^9$
$s_3 = \alpha^7$
Error-locator polynomial $\Lambda(x) = \alpha^{12}x^3 + 1 = (\alpha^{14}x - 1)(\alpha^9 x - 1)(\alpha^4 x - 1)$ is of degree 3, which means, that there occured 3 errors. To find error values, following set of equations must be solved:

$$\begin{bmatrix} \alpha^{14} & \alpha^9 & \alpha^4 \\ (\alpha^{14})^2 & (\alpha^9)^2 & (\alpha^4)^2 \\ (\alpha^{14})^3 & (\alpha^9)^3 & (\alpha^4)^3 \end{bmatrix} \begin{bmatrix} e_{k_1} \\ e_{k_2} \\ e_{k_3} \end{bmatrix} = \begin{bmatrix} \alpha^6 \\ \alpha^9 \\ \alpha^7 \end{bmatrix}$$

With use of Gaussian elimination, there can be found error values:

$$\begin{bmatrix} \alpha^{14} & \alpha^{9} & \alpha^{4} & \alpha^{6} \\ (\alpha^{14})^2 & (\alpha^{9})^2 & (\alpha^{4})^2 & \alpha^{9} \\ (\alpha^{14})^3 & (\alpha^{9})^3 & (\alpha^{4})^3 & \alpha^{7} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & \alpha \\ 0 & 1 & 0 & \alpha^2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

The results of set of equations are: $e_{k_1} = \alpha$, $e_{k_2} = \alpha^2$, $e_{k_3} = 1$. Error vector is of following form:

$$e(x) = e_{k_1} x^{\log_\alpha X_1} + e_{k_2} x^{\log_\alpha X_2} + e_{k_3} x^{\log_\alpha X_3} = \alpha x^{14} + \alpha^2 x^9 + x^4$$

4.3.1 Forney algorithm

Forney algorithm is used to find error values. Error value can be computed with use of following equation [14]:

$$e_{k_j} = -X_j \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})} \tag{4.10}$$

Error evaluator polynomial is computed as follows:

$$\Omega(x) \bmod x^{2t+1} = \Lambda(x) S(x)$$

Polynomial $\Lambda'(x)$ is a formal derivative of $\Lambda(x)$ and it is computed as follows:

$$\Lambda'(x) = \sum_{j=1}^{t} \Lambda_j x^{j-1}$$

Example 4.14.
Given is syndrome polynomial:

$$S(x) = \alpha^3 x^8 + x^7 + \alpha^4 x^6 + \alpha^6 x^5 + \alpha^3 x^4 + \alpha^7 x^3 + \alpha^9 x^2 + \alpha^6 x$$

Given is error-locator polynomial:

$$\Lambda(x) = \alpha^{12} x^3 + 1 = (\alpha^{14} x - 1)(\alpha^9 x - 1)(\alpha^4 x - 1)$$

Formal derivative of error-locator polynomial is expressed as:

$$\Lambda'(x) = \alpha^{12} x^2$$

$RS(15,7)$ code is used, so error correction capability is $t = 4$. Polynomial $\Omega(x)$ is as follows:

$$\Omega(x) \bmod x^{2t+1} = \Lambda(x) S(x)$$

$$\Omega(x) \bmod x^9 = (\alpha^{12}x^3+1)(\alpha^3x^8+x^7+\alpha^4x^6+\alpha^6x^5+\alpha^3x^4+\alpha^7x^3+\alpha^9x^2+\alpha^6x)$$
$$\Omega(x) \bmod x^9 = x^{11}+\alpha^{12}x^{10}+\alpha x^9+\alpha^7x^3+\alpha^9x^2+\alpha^6x$$
$$\Omega(x) = \alpha^7x^3+\alpha^9x^2+\alpha^6x$$

Errors are computed:

$$e_{k_1} = -X_1\frac{\Omega(X_1^{-1})}{\Lambda'(X_1^{-1})} = -\alpha^{14}\frac{\Omega(\alpha)}{\Lambda'(\alpha)} = \alpha$$

$$e_{k_2} = -X_2\frac{\Omega(X_2^{-1})}{\Lambda'(X_2^{-1})} = -\alpha^{9}\frac{\Omega(\alpha^6)}{\Lambda'(\alpha^6)} = \alpha^2$$

$$e_{k_3} = -X_3\frac{\Omega(X_3^{-1})}{\Lambda'(X_3^{-1})} = -\alpha^{4}\frac{\Omega(\alpha^{11})}{\Lambda'(\alpha^{11})} = 1$$

Error polynomial is as follows:

$$e(x) = e_{k_1}x^{\log_\alpha X_1}+e_{k_2}x^{\log_\alpha X_2}+e_{k_3}x^{\log_\alpha X_3} = \alpha x^{14}+\alpha^2x^9+x^4$$

4.4 Erasure decoding

Erasure decoding gives the opprtunity to increase number of erroneous positions which can be corrected [12] [14]. There is added new symbol which is called erasure. This symbol denotes that receiver couldn't recognize received symbol and marked it as erasure. Code with error correction capability t can correct following number of errors v and erasures f:

$$v + \frac{f}{2} \leq t$$

Erasures give additional information before decoding start – it is known what are erroneous positions. Given is following polynomial:

$$\Gamma(x) = \prod_{j=1}^{f}(1 - Y_jx),$$

where f denotes number of erasures in received vector, and Y_j denotes jth position of erasure. This polynomial is called erasure locator polynomial. Modified syndrome polynomial for erasures is as follows:

$$\Theta(x) = \Gamma(x)S(x) \bmod x^{2t+1}$$

Key equation for erasures is as follows:

$$\Lambda(x)\Theta(x) = \Omega(x) \bmod x^{2t+1}$$

Sugiyama algorithm can be used to solve this equation. Values of syndromes are computed in such way, that erased positions are filled with zeroes.

Modified Sugiyama algorithm for erasures is as follows:

1. Set variables $s(x) = x^{2t+1}$, $t(x) = \Theta(x)$, $A(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
2. Check if degree of $t(x)$ is less or equal than error correction capability t:
   - If
     $$\deg t(x) \leq t + \frac{f}{2} \text{ for } f \text{ even}$$
     lub
     $$\deg t(x) \leq t + \frac{f-1}{2} \text{ for } f \text{ odd,}$$
     then finish. The result is:
     $$\Lambda(x) = (A_{2,2}(0))^{-1}A_{2,2}(x)$$
   - Else continue.
3. Compute:
   $$Q(x) = \frac{s(x)}{t(x)}$$
4. Compute:
   $$\begin{bmatrix} s(x) \\ t(x) \end{bmatrix} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} \begin{bmatrix} s(x) \\ t(x) \end{bmatrix}$$
5. Compute:
   $$A(x) \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -Q(x) \end{bmatrix} A(x)$$
6. Return to point 2.

After calculation of error positions, error and erasure values have to be found. There is given following polynomial:

$$\Phi(x) = \Lambda(x)\Gamma(x)$$

This polynomial is called error/erasure locator polynomial. This polynomial can be used in modified Forney algorithm to find error and erasure values.

Modified Forney algorithm is as follows:

$$e_{k_j} = -X_j \frac{\Omega(X_j^{-1})}{\Phi'(X_j^{-1})}$$

$$f_{k_j} = -Y_j \frac{\Omega(Y_j^{-1})}{\Phi'(Y_j^{-1})}$$

Example 4.15.
Codeword of $RS(15,7)$ code is as follows:

$$\begin{aligned} c(x) = \alpha^6 x^{14} + \alpha x^{13} + x^{12} + \alpha^2 x^{11} + \alpha x^{10} + \alpha x^9 \\ + \alpha^2 x^8 + \alpha^3 x^7 + \alpha^7 x^6 + \alpha^9 x^5 + \alpha^5 x^4 + \alpha^9 x^3 + x + \alpha^{10} \end{aligned}$$

Error vector is as follows:

$$e(x) = \alpha x^{14} + \alpha^2 x^9 + x^4$$

Erasures are marked on positions 10 and 1, so $Y_1 = \alpha^{10}$ i $Y_2 = \alpha$. Received vector is:

$$\begin{aligned} r(x) = \alpha^{11} x^{14} + \alpha x^{13} + x^{12} + \alpha^2 x^{11} + f_{k_1} x^{10} + \alpha^5 x^9 \\ + \alpha^2 x^8 + \alpha^3 x^7 + \alpha^7 x^6 + \alpha^9 x^5 + \alpha^{10} x^4 + \alpha^9 x^3 + f_{k_2} x + \alpha^{10} \end{aligned}$$

Code can correct up to $t = 4$ errors. There occured $v = 3$ errors and $f = 2$ erasures. It is satisfied that:

$$\begin{gathered} v + \frac{f}{2} \leq t \\ 3 + 1 \leq 4 \end{gathered}$$

Erasure locator polynomial is as follows:

$$\Gamma(x) = \prod_{j=1}^{f} (1 - Y_j x) = (1 - \alpha^{10} x)(1 - \alpha x) = \alpha^{11} x^2 + \alpha^8 x + 1$$

Erasure positions are filled with zeroes:

$$\begin{aligned} r(x) = \alpha^{11} x^{14} + \alpha x^{13} + x^{12} + \alpha^2 x^{11} + 0 \cdot x^{10} + \alpha^5 x^9 \\ + \alpha^2 x^8 + \alpha^3 x^7 + \alpha^7 x^6 + \alpha^9 x^5 + \alpha^{10} x^4 + \alpha^9 x^3 + 0 \cdot x + \alpha^{10} \end{aligned}$$

Computed syndromes are as follows:
$s_1 = 0$
$s_2 = \alpha$
$s_3 = 1$
$s_4 = \alpha^8$
$s_5 = \alpha^5$
$s_6 = \alpha^{13}$
$s_7 = \alpha^2$

$s_8 = 1$
Modified syndrome polynomial is as follows:

$$\Theta(x) = \Gamma(x)S(x) \bmod x^{2t+1}$$

$$\Theta(x) = (\alpha^{11}x^2+\alpha^8x+1)(x^8 + \alpha^2x^7 + \alpha^{13}x^6 + \alpha^5x^5 + \alpha^8x^4 + x^3 + \alpha x^2) \bmod x^9$$

$$\Theta(x) = \alpha^6x^8 + \alpha^9x^7 + \alpha^4x^6 + \alpha^9x^5 + \alpha^{12}x^4 + \alpha^7x^3 + \alpha x^2$$

Error-locator polynomial is found by modified Sugiyama algorithm. Input data is:

$$s(x) = x^9 \qquad t(x) = \Theta(x)$$

Result of Sugiyama algorithm is:

$$\Lambda(x) = \alpha^{12}x^3 + 1 = (\alpha^{14}x - 1)(\alpha^9x - 1)(\alpha^4x - 1)$$

Error evaluator polynomial is as follows:

$$\Lambda(x)\Theta(x) = \Omega(x) \bmod x^{2t+1}$$

$$(\alpha^{12}x^3 + 1)(\alpha^6x^8 + \alpha^9x^7 + \alpha^4x^6 + \alpha^9x^5 + \alpha^{12}x^4 + \alpha^7x^3 + \alpha x^2) = \Omega(x) \bmod x^9$$

$$\Omega(x) = \alpha^{10}x^5 + \alpha^{12}x^4 + \alpha^7x^3 + \alpha x^2$$

Error/erasure locator polynomial is as follows:

$$\Phi(x) = \Lambda(x)\Gamma(x)$$

$$\Phi(x) = (\alpha^{12}x^3 + 1)(\alpha^{11}x^2 + \alpha^8x + 1)$$

$$\Phi(x) = (\alpha^{12}x^3 + 1)(\alpha^{11}x^2 + \alpha^8x + 1)$$

$$\Phi(x) = \alpha^8x^5 + \alpha^5x^4 + \alpha^{12}x^3 + \alpha^{11}x^2 + \alpha^8x + 1$$

Formal derivative of error/erasure locator polynomial is as follows:

$$\Phi'(x) = 5 \cdot \alpha^8x^4 + 4 \cdot \alpha^5x^3 + 3 \cdot \alpha^{12}x^2 + 2 \cdot \alpha^{11}x + 1 \cdot \alpha^8$$

$$\Phi'(x) = \alpha^8x^4 + \alpha^{12}x^2 + \alpha^8$$

Error values are as follows:

$$e_{k_1} = -X_1\frac{\Omega(X_1^{-1})}{\Phi'(X_1^{-1})} = -\alpha^{14}\frac{\Omega(\alpha)}{\Phi'(\alpha)} = \alpha$$

$$e_{k_2} = -X_2\frac{\Omega(X_2^{-1})}{\Phi'(X_2^{-1})} = -\alpha^9\frac{\Omega(\alpha^6)}{\Phi'(\alpha^6)} = \alpha^2$$

$$e_{k_3} = -X_3 \frac{\Omega(X_3^{-1})}{\Phi'(X_3^{-1})} = -\alpha^4 \frac{\Omega(\alpha^{11})}{\Phi'(\alpha^{11})} = 1$$

Erasure valus are as follows:

$$f_{k_1} = -Y_1 \frac{\Omega(Y_1^{-1})}{\Phi'(Y_1^{-1})} = -\alpha^{10} \frac{\Omega(\alpha^{5})}{\Phi'(\alpha^{5})} = \alpha$$

$$f_{k_2} = -Y_2 \frac{\Omega(Y_2^{-1})}{\Phi'(Y_2^{-1})} = -\alpha \frac{\Omega(\alpha^{14})}{\Phi'(\alpha^{14})} = 1$$

Received vector is corrected as follows:

$$c(x) = r(x) - (e(x) + f(x)) = r(x) - ((\alpha x^{14} + \alpha^2 x^9 + x^4) + (\alpha x^{10} + x))$$

Polynomial $f(x)$ denotes erasure polynomial, which is used to correct received vector.

4.5 Guruswami-Sudan algorithm

Algorithms which were described in sections 4.2.1, 4.2.3, 4.2.2 can correct errors, which number is not bigger than error correction capability of code. Error capability of code is equal to $t = \frac{d_{min}-1}{2}$. Decoding can be described as searching codeword in sphere surrounding received vector of radius t. If number of errors is not greater than t, then in this sphere is placed only one codeword. Increasing radius of this sphere and searching codeword in new sphere, it may happen, that there will be found two or more codewords and probably only one of them is original codeword. Such decoding is called list decoding and it lets to decode received vectors above limit of error correction capability t [7] [17] [12].

Guruswami-Sudan algorithm is used to decode codewords created by original method (3.2.1). Codeword is created by evaluating information polynomial for elements in support set, which is set of nonzero elements of field used to create code. Given is some information polynomial of degree less than k for code $RS(n, k)$. Codeword is created as follows:

$$c = (m(\alpha_1), m(\alpha_2), \dots, m(\alpha_n))$$

Codeword can be expressed also as ordered set of points:

$$c = ((\alpha_1, m(\alpha_1)), (\alpha_2, m(\alpha_2)), \dots, (\alpha_n, m(\alpha_n)))$$

Received vector from communication channel can be expressed as follows:

$$r = ((\alpha_1, r_0), (\alpha_2, r_1), \dots, (\alpha_n, r_{n-1}))$$

If on i-th position $m(\alpha_i) \neq r_{i-1}$, then it means that error occured on this position.

Guruswami-Sudana algorithm finds such set of polynomial $p(x)$, which can transform received vector r into codeword which is in Hamming distance less or equal to given error correction capability of algorithm. Guruswami-Sudan algorithm consists of two main steps:

- Interpolation step: First step of decoding is to construct polynomial of two variables:
$$Q(x,y) = \sum_i \sum_j a_{i,j} x^i x^j,$$
where zeroes of this polynomial are points of received vector (α_i, r_{i-1}), $Q(\alpha_i, r_{i-1}) = 0$. Polynomial $Q(x,y)$ is of degree $(1, k-1)$ of the smallest possible. To construct polynomial $Q(x,y)$ there is used parameter called multiplicity of zeroes m, which denotes the number of times where points (α_i, r_{i-1}) are zeroes of polynomial $Q(x,y)$ or how many times polynomial $Q(x,y)$ goes through points (α_i, r_{i-1}).
- Factorization step: Second step of decoding is factorization of polynomial $Q(x,y)$ to polynomials of type $y - p_i(x)$, where $p_i(x)$ is of degree less than k. One of polynomials $p_i(x)$ is probably original information polynomial which is used to create codeword.

To work with Guruswami-Sudan algorithm, there must be presented basic mathematical methods, which are used with algorithm.

Weighted degree

Guruswami-Sudan algorithm employs monomials of type $x^i y^j$. During decoding process there is proceeded comparision of such monomials. One of the methods to solve this problem is introduction of (u,v)-weighted degree (w-degree), where u and v are some positive integer numbers and they are not both zeroes:
$$\deg_w x^i y^j = ui + vj$$
Sometimes it may happen, that for two different monomials $x^{i_1}y^{j_1}$ and $x^{i_2}y^{j_2}$ values of weighted degree are the same. In order to solve such problem, there is introduced lexicographical order (w-lex) and reverse lexicographical order (w-revlex), where:
$$\begin{aligned} x^{i_1}y^{j_1} < x^{i_2}y^{j_2}, &\text{ for } ui_1 + vj_1 < ui_2 + vj_2 \\ &\text{or } ui_1 + vj_1 = ui_2 + vj_2 \text{ and } \begin{cases} i_1 < i_2 & \text{for w-lex} \\ i_1 > i_2 & \text{for w-revlex} \end{cases} \end{aligned}$$

Weighted degree of polynomial $Q(x, y)$ which consists of monomials $a_n x^{i_n} y^{j_n}$: $a_n \neq 0$ is the biggest degree of n-th monomial:

$$\deg_w Q(x, y) = \max\{\deg_w a_n x^{i_n} y^{j_n} : a_n \neq 0\}$$

When given is some (u, v) weighted degree, there can be created ordered list of monomials by using w-lex or w-revlex order. For some monomial $x^{i_n} y^{j_n}$ its order number on list is denoted as $Ind(x^{i_n} y^{j_n})$.

Example 4.16.
Given is $(u, v) = (1, 2)$ weighted degree. For exemplary monomials, their weighted degree is as follows:

$$\deg_w xy = \deg_w x^1 y^1 = 1 \cdot 1 + 2 \cdot 1 = 3$$

$$\deg_w x^2 y = 1 \cdot 2 + 2 \cdot 1 = 4$$

$$\deg_w x^3 = 1 \cdot 3 + 2 \cdot 0 = 3$$

It can be observed that $\deg_w xy = \deg_w x^3$. For w-lex order:

$$xy < x^3,$$

because looking at exponents of variable x, $1 < 3$. For w-revlex order:

$$xy > x^3,$$

because $1 \not< 3$.

Fragment of ordered list for $(1, 2)$ w-revlex is shown in table 4.3.

Multiplicities of zeroes

Multiplicity of zero in point $(0, 0)$ is such number m, for which a polynomial $Q(x, y)$ cannot be factorized to any polynomial of type $a_{i,j} x^i y^j$, where $a_{i,j} \neq 0$, że $i + j < m$. For point (α, β) polynomial $Q(x, y)$ polynomial $Q(x, y)$ is of multiplicity of zeroes of order m, if polynomial $Q(x + \alpha, y + \beta)$ is of multiplicity of zeroes of order m in point $(0, 0)$. To compute what multiplicity of zeroes is given polynomial $Q(x, y)$, there can be used Hasse derivative. Equation for Hasse derivative is as follows:

$$Q_{r,s}(x, y) = \sum_i \sum_j \binom{i}{r} \binom{j}{s} a_{i,j} x^{i-r} y^{j-s}$$

$x^{i_n}y^{j_n}$	$\deg_w x^{i_n}y^{j_n}$	$Ind(x^{i_n}y^{j_n})$
x^0y^0	0	0
x^1y^0	1	1
x^2y^0	2	2
x^0y^1	2	3
x^3y^0	3	4
x^1y^1	3	5
x^4y^0	4	6
x^2y^1	4	7
x^0y^2	4	8
...	...	...

Table 4.3: Fragment of ordered list of monomials of type $x^{i_n}y^{j_n}$ for order $(1, 2)$ w-revlex.

For binomial coefficient $\binom{n}{k}$ it is true that $n \geq k \geq 0$, then Hasse derivative can be expressed as:

$$Q_{r,s}(x,y) = \sum_{i\geq r}\sum_{j\geq s}\binom{i}{r}\binom{j}{s}a_{i,j}x^{i-r}y^{j-s}$$

Some polynomial $Q(x, y)$ is of multiplicity of zeroes of order m in points (α, β), if:

$$Q_{r,s}(\alpha,\beta) = 0 \qquad \text{for } 0 \leq r+s < m$$

Example 4.17.
Given is polynomial $Q(x, y) = xy + x^2y^3$. Order of multiplicity of zeroes in point $(0, 0)$ is $m = 2$, because the smallest sum of exponents is the smallest for nonzero monomial xy and is equal to 2.

Hasse derivative for polynomial $Q(x, y)$ is expressed as:

$$Q_{r,s}(x,y) = \binom{1}{r}\binom{1}{s}x^{1-r}y^{1-s} + \binom{2}{r}\binom{3}{s}x^{2-r}y^{3-s}$$

If order of multiplicity of zeroes is $m = 2$, then it must be satisfied for $0 \leq r+s < 2$ that $Q_{r,s}(0, 0) = 0$. Such (r, s) pairs are $(0, 0)$, $(0, 1)$ and $(1, 0)$:

$$Q_{0,0}(0,0) = \binom{1}{0}\binom{1}{0}0^{1-0}0^{1-0} + \binom{2}{0}\binom{3}{0}0^{2-0}0^{3-0} = 0$$

$$Q_{0,1}(0,0) = \binom{1}{0}\binom{1}{1}0^{1-0}0^{1-1} + \binom{2}{0}\binom{3}{1}0^{2-0}0^{3-1} = 0$$

$$Q_{1,0}(0,0) = \binom{1}{1}\binom{1}{0}0^{1-1}0^{1-0} + \binom{2}{1}\binom{3}{0}0^{2-1}0^{3-0} = 0$$

However for $(r,s) = (1,1)$, where $0 \leq 2 \not< 2$ and if $0^0 = 1$, then:

$$Q_{1,1}(0,0) = \binom{1}{1}\binom{1}{1}0^{1-1}0^{1-1} + \binom{2}{1}\binom{3}{0}0^{2-1}0^{3-1} = 1$$

Number of polynomial coefficients

Interpolation step of Guruswami-Sudan algorithm produces following polynomial of two variables:

$$Q(x,y) = \sum_i \sum_j a_{i,j} x^i y^j$$

This polynomial can be expressed also as:

$$Q(x,y) = \sum_{i=0}^{C} a_i \phi_i(x,y),$$

where $\phi_i(x,y)$ denotes i-th monomial $x^i y^j$ in given order (w-lex or w-revlex). Parameter C can be expressed as follows:

$$C = n\binom{m+1}{2}$$

It is true that some polynomial $Q(x,y)$ goes through point (x_i, y_i) m times, if Hasse derivative $Q_{r,s}(x,y)$ for points (x_i,y_i) is equal to $Q_{r,s}(x_i,y_i) = 0$ for all pairs $0 \leq r+s < m$. The number of such pairs (r,s) can expressed as:

$$\binom{m+1}{2}$$

For code $RS(n,k)$ there n points, through which goes polynomial $Q(x,y)$, so number of pairs (r,s) for n points is expressed as:

$$n\binom{m+1}{2}$$

Example 4.18.
Given is polynomial $Q(x,y) = xy + x^2y^3$, which is of multiplicity of zeroes in point $(0,0)$ equal to $m = 2$. In this case all monomials of type $a_{i,j}x^iy^j$, where $i + j < 2$, have coefficients $a_{i,j} = 0$. It can be observed that for $a_{0,0}x^0y^0$,

$a_{0,1}x^0y^1$ and $a_{1,0}x^1y^0$, the coefficient is zero. The number of these monomials can be expressed as:

$$\binom{m+1}{2} = \binom{2+1}{2} = \binom{3}{2} = 3$$

Polynomial goes through point $(0,0)$ $m = 2$ times. This only one point must be passed, so:

$$C = 1 \cdot \binom{3}{2} = 3$$

Minimal polynomial, which goes through $(0,0)$ $m = 2$ times is as follows:

$$Q(x,y) = \sum_{i=0}^{C} a_i \phi_i(x,y)$$

$$Q(x,y) = 0 \cdot x^0y^0 + 0 \cdot x^1y^0 + 0 \cdot x^0y^1 + a_{1,1} \cdot x^1y^1$$

or

$$Q(x,y) = 0 \cdot x^0y^0 + 0 \cdot x^1y^0 + 0 \cdot x^0y^1 + a_{2,0} \cdot x^2y^0$$

or

$$Q(x,y) = 0 \cdot x^0y^0 + 0 \cdot x^1y^0 + 0 \cdot x^0y^1 + a_{0,2} \cdot x^0y^2$$

Coefficients $a_{1,1}$, $a_{2,0}$, $a_{0,2}$ are not equal to zero.

Error correction capability

Error correction capability of Guruswami-Sudan algorithm for code $RS(n,k)$ depends on length of codeword n, number of information elements k and parameter m, which is order of multiplicity of zeroes. Weighted degree for code $RS(n,k)$ is set to $(1, k-1)$. Weighted degree and w-lex order settles order of monomials of type x^iy^j. During writing this master thesis, there was used w-revlex order. Error correction capability of algorithm for parameter m is as follows:

$$t_m = n - 1 - \lfloor \frac{\{\max K : Ind(x^K) \leq C\}}{m} \rfloor$$

Example 4.19.
Given is code $RS(7,3)$. There is used $(1, k-1)$ w-revlex order, which is equal to $(1,2)$ w-revlex. Error correction capability of Guruswami-Sudan

algorithm depends on parameter m.
For $m = 2$:

$$C = n\binom{m+1}{2} = 7 \cdot \binom{3}{2} = 21$$

$$t_m = n - 1 - \lfloor\frac{\{\max K : Ind(x^K) \leq C\}}{m}\rfloor$$

$$t_2 = 7 - 1 - \lfloor\frac{\{\max K : Ind(x^K) \leq 21\}}{2}\rfloor$$

On the ordered list(if there were shown more elements on table 4.3) $Ind(x^8) = 20$ and this value is greatest for monomial of type x^K, where $Ind(x^K) \leq 21$, so:

$$t_2 = 6 - \lfloor\frac{8}{2}\rfloor = 2$$

For $m = 3$:

$$C = 7\binom{3+1}{2} = 42$$

$$t_3 = 7 - 1 - \lfloor\frac{\{\max K : Ind(x^K) \leq 42\}}{3}\rfloor$$

For monomial x^{12} it is true that $Ind(x^{12}) = 42$.

$$t_3 = 6 - \lfloor\frac{12}{3}\rfloor = 2$$

For $m = 4$:

$$C = 7\binom{4+1}{2} = 70$$

$$t_4 = 7 - 1 - \lfloor\frac{\{\max K : Ind(x^K) \leq 70\}}{4}\rfloor$$

For monomial x^{15} it is true that $Ind(x^{15}) = 64$.

$$t_4 = 6 - \lfloor\frac{15}{4}\rfloor = 6 - 3 = 3$$

For parameter $m = 4$ algorithm can correct 1 more errors than it is denoted by error correction capability of code which is $t = 2$.

List of results

Guruswami-Sudan algorithm returns list of polynomials, where probably is placed original information polynomial. Maximal number of returned elements is as follows:

$$L_m = \{\max K : Ind(y^K) \leq C\}$$

Example 4.20.
For code $RS(7,3)$, order $(1,2)$ w-revlex, parameter $m = 4$, maximal number of returned elements is as follows:

$$L_4 = \{\max K : Ind(y^K) \leq 70\} = 7,$$

because for monomial y^7 it is true that $Ind(y^7) = 63$ and it is greatest for y^K, where $Ind(y^K) \leq 70$.

4.5.1 Kötter algorithm

Kötter algorithm computes Hasse derivative $Q_{r,s}(x,y)$ for all points of received vector. On of the key element of algorithm is order of pairs (r,s), which are parameters of Hasse derivative. This order is as follows:

$$(0,0),(0,1),\ldots,(0,m-1),(1,0),(1,1),\ldots,(1,m-2),\ldots,(m-1,0)$$

It can be also expressed as:

$$K = \{(0,0),(0,1),\ldots,(0,m-1),(1,0),\ldots,(m-1,m-1)\} \backslash \{(i,j) : i+j < m\}$$

Kötter algorithm is used to construct polynomial $Q(x,y)$ during interpolation step of Guruswami-Sudan algorithm and is as follows:

1. Given is following input data:
   - maximal number of returned elements on the list L_m,
   - set of n points (α_i, β_i) for $i = \{1,2,\ldots,n\}$, which symbolize received vector,
   - order of multiplicities of zeroes for points (α_i, β_i) – m,
   - order of type $(1, k-1)$ w-revlex for code $RS(n,k)$,
   - ordered table of pairs (r,s) – K.
2. Create $L_m + 1$ polynomials $g_j(x,y) = y^j$, where $j = \{0,1,\ldots,L_m\}$.
3. Set variable $i = 1$, which will point number of currently evaluated point (α_i, β_i).
4. If $i > n$, then go to point 14, else set new point $(a,b) = (\alpha_i, \beta_i)$ and continue.

5. Set variable $j = 1$, which will point (r, s) pair in table K.
6. If $j > \binom{m+1}{2}$, then increase variable $i = i + 1$ and return to point 4, else set current pair of Hasse derivative arguments $(r, s) = K[j]$.
7. For each polynomial $g_j(x, y)$, where $j = \{0, 1, \ldots, L_m\}$, compute discrepancy Δ_j, which will be used to modify polynomial $g_j(x, y)$:

$$\Delta_j = Q_{r,s} g_j(a, b)$$

8. Construct set J, which consists of indices j, where $\Delta_j \neq 0$.
9. If set J is empty, then increase $j = j + 1$ and go to point 6, else continue.
10. Find polynomial of least degree in set J and remember its index j^*:

$$j^* = \arg\min\{g_j(x, y) : j \in J\}$$

11. Set two auxiliary variables:

$$f = g_{j^*}(x, y) \qquad \Delta = \Delta_{j^*}$$

12. For each polynomial $g_j(x, y)$, where $j \in J$, do:
    - if $j \neq j^*$, compute:

    $$g_j(x, y) = \Delta g_j(x, y) - \Delta_j f$$

    - if $j = j^*$, compute:

    $$g_j(x, y) = \Delta(x - a) f$$

13. Increase variable $j = j + 1$ and go to point 6.
14. Choose polynomial of least degree within set $g_j(x, y)$ and return it:

$$Q(x, y) = g_j(x, y),$$

where $\deg_w g_j(x, y)$ is least for $j = \{0, 1, \ldots, L_m\}$.

Example 4.21.
Given is code $RS(7, 3)$, parameter $m = 4$. Algorithm can correct up to $t_4 = 3$ errors and returns maximally $L_4 = 7$ potential information polynomials. Given is vector of information elements:

$$m = (\alpha, \alpha^2, 1)$$

Codeword created with original method is as follows:

$$c = (\alpha^5, \alpha^6, \alpha, 0, \alpha^6, 0, \alpha^5)$$

Received vector is as follows:

$$r = (\alpha^5, \alpha^6, \alpha, 0, \alpha^2, 1, \alpha^4)$$

Error vector is as follows:

$$e = (0, 0, 0, 0, 1, 1, 1)$$

There occured 3 errors. Previously described algorithms (PGZ, BMA etc.) are able to correct up to 2 errors. There will be created polynomial $Q(x, y)$, which will be $m = 4$ times go through points:

$$(1, \alpha^5), (\alpha, \alpha^6), (\alpha^2, \alpha), (\alpha^3, 0), (\alpha^4, \alpha^5), (\alpha^5, 1), (\alpha^6, \alpha^4)$$

First point comes from support set, and second point comes from received vector. Parameter m is equal to 4, so parameters (r, s) of Hasse derivative are:

$$K = \{(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (3, 0)\}$$

For code $RS(7, 3)$ there is used $(1, k-1) = (1, 2)$ w-revlex order.

First step of Kötter algorithm is to created $L_m + 1$ polynomials $g_j(x, y)$:

$$g_0(x, y) = 1 \quad g_1(x, y) = y \quad g_2(x, y) = y^2$$

$$g_3(x, y) = y^3 \quad g_4(x, y) = y^4 \quad g_5(x, y) = y^5$$

$$g_6(x, y) = y^6 \quad g_7(x, y) = y^7$$

Variable i is set to $i = 1$. Evaluated point is:

$$(a, b) = (\alpha_i, \beta_i) = (\alpha_1, \beta_1) = (1, \alpha^5)$$

Variable j is set to $j = 1$ and points on first pair (r, s) from table K: $(r, s) = K[1] = (0, 0)$ For each polynomial $g_j(x, y)$ there is computed Hasse derivative with parameters $(r, s) = (0, 0)$ and for point
$(x, y) = (a, b) = (1, \alpha^5)$:

$$\Delta_j = (g_j)_{r,s}(x, y) = \sum_{i \geq r} \sum_{j \geq s} \binom{i}{r} \binom{j}{s} a_{i,j} x^{i-r} y^{j-s}$$

For $g_0(x, y) = 1$:

$$\Delta_0 = (g_0)_{0,0}(1, \alpha^5) = \binom{0}{0} \binom{0}{0} 1^{0-0} (\alpha^5)^{0-0} = 1$$

For $g_1(x,y) = y$:

$$\Delta_1 = (g_1)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{1}{0}1^{0-0}(\alpha^5)^{1-0} = \alpha^5$$

For $g_2(x,y) = y^2$:

$$\Delta_2 = (g_2)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{2}{0}1^{0-0}(\alpha^5)^{2-0} = \alpha^3$$

For $g_3(x,y) = y^3$:

$$\Delta_3 = (g_3)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{3}{0}1^{0-0}(\alpha^5)^{3-0} = \alpha$$

For $g_4(x,y) = y^4$:

$$\Delta_4 = (g_4)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{4}{0}1^{0-0}(\alpha^5)^{4-0} = \alpha^6$$

For $g_5(x,y) = y^5$:

$$\Delta_5 = (g_5)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{5}{0}1^{0-0}(\alpha^5)^{5-0} = \alpha^4$$

For $g_6(x,y) = y^6$:

$$\Delta_6 = (g_6)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{6}{0}1^{0-0}(\alpha^5)^{6-0} = \alpha^2$$

For $g_7(x,y) = y^7$:

$$\Delta_7 = (g_7)_{0,0}(1,\alpha^5) = \binom{0}{0}\binom{7}{0}1^{0-0}(\alpha^5)^{7-0} = 1$$

All values of Δ_j are not equal to zero, so set J is as follows:

$$J = \{0,1,2,3,4,5,6,7\}$$

The least weighted degree $\deg_w$ is $g_0(x,y) = 1$, so:

$$j^* = 0$$

$$f = g_0(x,y) = 1$$

$$\Delta = \Delta_0 = 1$$

Polynomials $g_j(x,y)$ are modified. For $j = j^* = 0$:

$$g_0(x,y) \leftarrow \Delta(x-a)f = 1 \cdot (x-1) \cdot 1 = x+1$$

For $j \neq j^*$:

$$g_1(x,y) \leftarrow \Delta g_1(x,y) - \Delta_1 f = 1 \cdot y - \alpha^5 \cdot 1 = y + \alpha^5$$

$$\begin{aligned}
g_2(x,y) &\leftarrow \Delta g_2(x,y) - \Delta_2 f = 1 \cdot y^2 - \alpha^3 \cdot 1 = y^2 + \alpha^3 \\
g_3(x,y) &\leftarrow \Delta g_3(x,y) - \Delta_3 f = 1 \cdot y^3 - \alpha \cdot 1 = y^3 + \alpha \\
g_4(x,y) &\leftarrow \Delta g_4(x,y) - \Delta_4 f = 1 \cdot y^4 - \alpha^6 \cdot 1 = y^4 + \alpha^6 \\
g_5(x,y) &\leftarrow \Delta g_5(x,y) - \Delta_5 f = 1 \cdot y^5 - \alpha^4 \cdot 1 = y^5 + \alpha^4 \\
g_6(x,y) &\leftarrow \Delta g_6(x,y) - \Delta_6 f = 1 \cdot y^6 - \alpha^2 \cdot 1 = y^6 + \alpha^2 \\
g_7(x,y) &\leftarrow \Delta g_7(x,y) - \Delta_7 f = 1 \cdot y^7 - 1 \cdot 1 = y^7 + 1
\end{aligned}$$

Next, there is increased variable $j \leftarrow j+1$ and algorithm modifies polynomials $g_j(x,y)$ for next pair (r,s). When there have been evaluated all points, then from polynomials $g_j(x,y)$ must be choosen this of least weighted degree. The result of algorithm is as follows:

$$\begin{aligned}
Q(x,y) = {} & y^7(\alpha^2) + y^6(\alpha^5x^3 + \alpha x^2 + \alpha^6 x + \alpha^6) \\
& + y^5(\alpha x^5 + \alpha^6x^4 + \alpha^4x^3 + \alpha^4x^2 + \alpha^2) \\
& + y^4(x^7 + \alpha^3x^6 + \alpha x^5 + x^4 + \alpha^2x^3 + \alpha^5x + \alpha^2) \\
& + y^3(\alpha x^9 + \alpha^6x^8 + \alpha^6x^7 + \alpha^4x^6 + \alpha^6x^5 + \alpha^2x^4 \\
& \qquad + \alpha^3x^3 + \alpha^3x^2 + \alpha^3x + \alpha^2) \\
& + y^2(\alpha^2x^{11} + \alpha^4x^{10} + \alpha^4x^9 + \alpha x^8 + \alpha^5x^7 + \alpha^4x^6 \\
& \qquad + x^5 + \alpha^2x^4 + x^3 + \alpha^4x^2 + x + \alpha^3) \\
& + y(\alpha x^{13} + \alpha^6x^{12} + \alpha^6x^{11} + x^9 + \alpha^4x^8 + \alpha^5x^7 + \alpha^2x^5 \\
& \qquad + \alpha^6x^4 + \alpha^5x^3 + \alpha^4x + \alpha^4) \\
& + (\alpha^6x^{14} + x^{13} + \alpha^3x^{12} + \alpha^6x^{11} + \alpha^3x^{10} \\
& \qquad + \alpha x^9 + \alpha^5x^8 + \alpha^4x^7 + \alpha x^6 + \alpha^4x^4 + x^2 + \alpha^2x)
\end{aligned}$$

Each points from received vector:

$$(1,\alpha^5), (\alpha,\alpha^6), (\alpha^2,\alpha), (\alpha^3,0), (\alpha^4,\alpha^5), (\alpha^5,1), (\alpha^6,\alpha^4),$$

is zero ofpolynomial $Q(x,y)$. Furthermore, each point is zero of multiplicity of order 4, which means that polynomial $Q(x,y)$ goes through points 4 times. Hasse derivative of polynomial $Q(x,y)$ for each points and all pairs $r+s<4$ is equal to zero.

4.5.2 Roth-Ruckenstein algorithm

Roth-Ruckenstein algorithm is used to factorize polynomial $Q(x, y)$ to polynomials $y - p_i(x)$, which divide $Q(x, y)$ [18]. It is possible to set maximaly degree of polynomials $p_i(x)$ and for code $RS(n, k)$ it is $k - 1$. Algorithm works recursively and searches in depth all possible solutions. Each recursive call is to find new coefficient of some of searched polynomial $p_i(x)$. Maximal depth of recursion is $k - 1$.
Roth-Ruckenstein algorithm is as follows:

1. Input data is as follows:
   - polynomial of two variables $Q(x, y)$, which is factorized to polynomials of type $y - p_i(x)$ of degree less or equal to $k - 1$,
   - maximal degree of polynomials $p_i(x)$, which is $k - 1$,
   - variable u which describes depth of recursion. For first call of algorithm $u = 0$ and for next calls this value is passed as parameter.
   - polynomial $p_i(x)$. For first call it is $p_i(x) = 0$, and for all next calls it is passed as parameter. On recursion depth $u = 0$ there are computed coefficients for x^0, on depth $u = 1$ for x^1 etc. For each new call of algorithm there is passed copy of polynomial $p_i(x)$, whose coefficients were computed in previous branch of recursion.
2. Find roots of polynomial $Q(0, y)$, substituting y with all elements of Galois field over which is created code.
3. For each root α_n of polynomial $Q(0, y)$ do:
   - compute:
     $$Q_i(x, y) \leftarrow Q(x, xy + \alpha_n)$$
   - find such monomial of type x^r for greatest exponent r, where x^r divides $Q_i(x, y)$ without remainder,
   - divide polynomial $Q_i(x, y)$ by monomial x^r:
     $$Q_i(x, y) \leftarrow \frac{Q_i(x, y)}{x^r}$$
   - set root α_i as coefficient of polynomial $p_i(x)$ for x^u,
     - if $(u + 1) < k$, then open new recursive call and go to point 2 with parameters of polynomial $Q_i(x, y)$, polynomial $p_i(x)$ and depth of recursion $u + 1$,
     - else if $(u + 1) = k$ and $Q_i(x, 0) = 0$, then polynomial $p_i(x)$ is returned as result and evaluated branch of recursion is closed,
     - else close currect branch of recursion.

Example 4.22.
There will be factorized following polynomial:

$$
\begin{aligned}
Q(x,y) = {} & y^7(\alpha^2) + y^6(\alpha^5x^3 + \alpha x^2 + \alpha^6 x + \alpha^6) \\
& + y^5(\alpha x^5 + \alpha^6x^4 + \alpha^4x^3 + \alpha^4x^2 + \alpha^2) \\
& + y^4(x^7 + \alpha^3x^6 + \alpha x^5 + x^4 + \alpha^2x^3 + \alpha^5x + \alpha^2) \\
& + y^3(\alpha x^9 + \alpha^6x^8 + \alpha^6x^7 + \alpha^4x^6 + \alpha^6x^5 + \alpha^2x^4 \\
& \qquad\qquad + \alpha^3x^3 + \alpha^3x^2 + \alpha^3x + \alpha^2) \\
& + y^2(\alpha^2x^{11} + \alpha^4x^{10} + \alpha^4x^9 + \alpha x^8 + \alpha^5x^7 + \alpha^4x^6 \\
& \qquad\qquad + x^5 + \alpha^2x^4 + x^3 + \alpha^4x^2 + x + \alpha^3) \\
& + y(\alpha x^{13} + \alpha^6x^{12} + \alpha^6x^{11} + x^9 + \alpha^4x^8 + \alpha^5x^7 + \alpha^2x^5 \\
& \qquad\qquad + \alpha^6x^4 + \alpha^5x^3 + \alpha^4x + \alpha^4) \\
& + (\alpha^6x^{14} + x^{13} + \alpha^3x^{12} + \alpha^6x^{11} + \alpha^3x^{10} \\
& \qquad + \alpha x^9 + \alpha^5x^8 + \alpha^4x^7 + \alpha x^6 + \alpha^4x^4 + x^2 + \alpha^2x)
\end{aligned}
$$

For code $RS(7,3)$ information polynomials can be of maximal degree of $k - 1 = 3 - 1 = 2$. Initial depth of recursion is $u = 0$. Firstly, there are searched roots of polynomial:

$$Q(0,y) = y^7(\alpha^2) + y^6(\alpha^6) + y^5(\alpha^2) + y^4(\alpha^2) + y^3(\alpha^2) + y^2(\alpha^3) + y(\alpha^4)$$

Following elements are roots of polynomial:

$$\alpha_1 = 0 \quad \alpha_2 = 1 \quad \alpha_3 = \alpha$$

$$\alpha_4 = \alpha^2 \quad \alpha_5 = \alpha^6$$

First there will be evaluated element $\alpha_1 = 0$. There is created polynomial:

$$Q_1(x,y) \leftarrow Q(x, xy + \alpha_1) = Q(x, xy + 0)$$

$$\begin{tikzcd}
& 0 \arrow[r] & \alpha^5 \arrow[r] & 0 & \\
& 1 \arrow[r] & 1 \arrow[r] & \alpha^2 & \\
Q\,(x,y) \arrow[ruu] \arrow[ru] \arrow[r] \arrow[rd] \arrow[rdd] & \alpha \arrow[r] & \alpha^2 \arrow[r] & 1 & \\
& \alpha^2 \arrow[r] & 0 \arrow[r] & \alpha^3 & \\
& \alpha^6 \arrow[r] & \alpha^5 \arrow[r] & \alpha^4 & \text{is not root}
\end{tikzcd}$$

Figure 4.2: Branches of recursion for Roth-Ruckenstein algorithm.

$$\begin{aligned}
Q_1(x,y) = {} & y^7(\alpha^2x^7) \\
& + y^6(\alpha^5x^9 + \alpha x^8 + \alpha^6x^7 + \alpha^6x^6) \\
& + y^5(\alpha x^{10} + \alpha^6x^9 + \alpha^4x^8 + \alpha^4x^7 + \alpha^2x^5) \\
& + y^4(x^{11} + \alpha^3x^{10} + \alpha x^9 + x^8 + \alpha^2x^7 + \alpha^5x^5 + \alpha^2x^4) \\
& + y^3(\alpha x^{12} + \alpha^6x^{11} + \alpha^6x^{10} + \alpha^4x^9 \\
& \qquad +\alpha^6x^8 + \alpha^2x^7 + \alpha^3x^6 + \alpha^3x^5 + \alpha^3x^4 + \alpha^2x^3) \\
& + y^2(\alpha^2x^{13} + \alpha^4x^{12} + \alpha^4x^{11} + \alpha x^{10} \\
& \qquad +\alpha^5x^9 + \alpha^4x^8 + x^7 + \alpha^2x^6 + x^5 + \alpha^4x^4 + x^3 + \alpha^3x^2) \\
& + y(\alpha x^{14} + \alpha^6x^{13} + \alpha^6x^{12} + x^{10} + \alpha^4x^9 \\
& \qquad +\alpha^5x^8 + \alpha^2x^6 + \alpha^6x^5 + \alpha^5x^4 + \alpha^4x^2 + \alpha^4x) \\
& + (\alpha^6x^{14} + x^{13} + \alpha^3x^{12} + \alpha^6x^{11} + \alpha^3x^{10} \\
& \qquad +\alpha x^9 + \alpha^5x^8 + \alpha^4x^7 + \alpha x^6 + \alpha^4x^4 + x^2 + \alpha^2x)
\end{aligned}$$

The greatest monomial of type x^r which divides $Q_1(x,y)$ withour remainder is x^1. Polynomial $Q_1(x,y)$ is divided by x^1:

$$Q_1(x,y) \leftarrow \frac{Q_1(x,y)}{x}$$

$$
\begin{aligned}
Q_1(x,y) = {} & y^7(\alpha^2x^6) \\
& + y^6(\alpha^5x^9 + \alpha x^7 + \alpha^6x^6 + \alpha^6x^5) \\
& + y^5(\alpha x^9 + \alpha^6x^8 + \alpha^4x^7 + \alpha^4x^6 + \alpha^2x^4) \\
& + y^4(x^{10} + \alpha^3x^9 + \alpha x^8 + x^7 + \alpha^2x^6 + \alpha^5x^4 + \alpha^2x^3) \\
& + y^3(\alpha x^{11} + \alpha^6x^{10} + \alpha^6x^9 + \alpha^4x^8 \\
& \qquad + \alpha^6x^7 + \alpha^2x^6 + \alpha^3x^5 + \alpha^3x^4 + \alpha^3x^3 + \alpha^2x^2) \\
& + y^2(\alpha^2x^{12} + \alpha^4x^{11} + \alpha^4x^{10} + \alpha x^9 \\
& \qquad + \alpha^5x^8 + \alpha^4x^7 + x^7 + \alpha^2x^5 + x^4 + \alpha^4x^3 + x^2 + \alpha^3x) \\
& + y(\alpha x^{13} + \alpha^6x^{12} + \alpha^6x^{11} + x^9 + \alpha^4x^8 \\
& \qquad + \alpha^5x^7 + \alpha^2x^5 + \alpha^6x^4 + \alpha^5x^3 + \alpha^4x + \alpha^4) \\
& + (\alpha^6x^{13} + x^{12} + \alpha^3x^{11} + \alpha^6x^{10} + \alpha^3x^9 \\
& \qquad + \alpha x^8 + \alpha^5x^7 + \alpha^4x^6 + \alpha x^5 + \alpha^4x^3 + x + \alpha^2)
\end{aligned}
$$

There is remembered root $\alpha_1 = 0$ and it becomes coefficient for x^0 of polynomial $p_1(x) = 0 \cdot x^0$. Then there is called new recursive procedure for $Q_1(x,y)$, recursion depth $u \leftarrow u+1$, polynomial $p_1(x)$. Subsequent steps of algorithm are shown on picture 4.2. There result of algorithms are following 4 polynomials:

$$
\begin{aligned}
p_1(x) &= \alpha^5x \\
p_2(x) &= \alpha^2x^2 + x + 1 \\
p_3(x) &= x^2 + \alpha^2x + \alpha \\
p_4(x) &= \alpha^3x^2 + \alpha^2
\end{aligned}
$$

Polynomial $\alpha^4x^2 + \alpha^5x + \alpha^6$ is not result of algorithm work, because in the last evaluated recursion element α^4 is not root of polynomial $Q_{5,1,1}(x,0)$. Returned polynomials creates following codewords of code $RS(7,3)$:

$$
\begin{aligned}
c_1 &= (\alpha^5, \alpha^6, 1, \alpha, \alpha^2, \alpha^3, \alpha^4) \\
c_2 &= (\alpha^2, \alpha^6, 0, 0, \alpha^2, 1, \alpha^6) \\
c_3 &= (\alpha^5, \alpha^6, \alpha, 0, \alpha^6, 0, \alpha^5) \\
c_4 &= (\alpha^5, \alpha^3, \alpha^6, 0, \alpha, 1, \alpha^4)
\end{aligned}
$$

Original received vector was:

$$r = (\alpha^5, \alpha^6, \alpha, 0, \alpha^2, 1, \alpha^4)$$

Hamming distance for vector r in order to each codeword c_i is equal to 3. Codeword c_3 is original codeword.

4.5.3 Kötter-Vardy algorithm

Kötter-Vardy algorithm is used to transform probabilistic reliability information to set of points which will next used to construct polynomial $Q(x, y)$ [19] [18]. To each point is assigned own order of multiplicity of zeroes m_b. Next, there is created set of points and it is used as input data for Guruswami-Sudan algorithm.
Köttera-Vardy algorithm is as follows:

1. Input data is as follows:
   - reliability matrix Π, where for code $RS(n, k)$ created over $GF(q)$, it consists of n columns and q rows. Each column describes particular element in received vector and in rows there is given probability which describes that given element is some element in $GF(q)$. For each column sum of probabilities is 1.
   - parameter s, which denotes sum of all orders of multiplicity of zeroes m_b, $s = \sum_{b=1}^{n} m_b$.
2. Create matrix M with the same number columns and rows as matrix Π, and set all values to zero.
3. Find in matrix Π an element with greatest value on some position (i, j), which means i-th row and j-th column.
4. Compute:
$$\Pi_{i,j} \leftarrow \frac{\Pi_{i,j}}{M_{i,j} + 2}$$
5. Compute:
$$M_{i,j} \leftarrow M_{i,j} + 1$$
6. Decrease s by 1, $s \leftarrow s - 1$. If $s = 0$ then return matrix M. Points which are used during interpolation are nonzero elements of matrix M on position (j, i), where j denotes number of column, and i number of row. Order of multiplicity of zeroes for given points is $m_b \leftarrow M_{j,i}$. If $s > 0$, then return to point 3.

Decoder which employs Kötter-Vardy algorithm, receives as input data received vector from communication channel. In the case of previously described algorithm it was hard information, which means that elements of vector were elements of Galois field. In Kötter-Vardy algorithm's case this is soft information, where for each element of received vector there is computed probability that element is some element in Galois field. During writing this master thesis, Galois field elements were represented as binary vectors, so they consists of elements 1 and 0. For soft decoding elements of Galois field are represented as vectors of rational numbers. They provide more information for soft decoding and sometimes it is possible to correct more errors than

for hard decoding. For master thesis' purpose, there was created a method for interfering received vector and for creating reliability matrix.
Algorithm for interfering codeword is as follows:

1. Input data:
   - codeword of $RS(n, k)$ code, where elements of vector are binary,
   - parameter d, which influences amplitude of noise,
   - parameter e, which influences frequency of noise.
2. Modulate codeword, that $1 \to -1$ and $0 \to 1$
3. For each bit of codeword do:
   - choose random value e_{rand} from 0 to 32767, using uniform probability distribution. If $e_{rand} < e$, compute:

     $$X = \frac{(e_{rand} \bmod d)}{1000},$$

     then, if e_{rand} is even then to evaluated bit add value X, and if e_{rand} is odd then from evaluated bit subtract value X.

The method to generate reliability matrix Π is as follows:

1. Input data:
   - received vector from communication channel, whose bits are represented as rational numbers,
   - parameter s, which is used to compute probability for some elements that it is one of elements in $GF(q)$.
2. For each element of received vector r_i, where $i = \{0, 1, \ldots, n-1\}$, do:
   - Compute Manhattan distance between vector of element of received vector and each element in $GF(p^m)$:

     $$d(r_i, \alpha_j) = \sum_{a=1}^{m} |r_i^{(a)} - \alpha_j^{(a)}|,$$

     where $r_i^{(a)}$ denotes a-th bit of vector r_i, α_j denotes j-th element of $GF(p^m)$, $\alpha_j^{(a)}$ denotes a-th bit of element α_j. Recall that $0 \to 1$ and $1 \to -1$.
   - next, for each computed value of distance do:

     $$d(r_i, \alpha_j) \leftarrow d(r_i, \alpha_j)^{-s}$$

   - compute, where $GF(p^m) = GF(q)$:

     $$d(r_i, \alpha_j) \leftarrow \frac{d(r_i, \alpha_j)}{\sum_{a=1}^{q} d(r_i, \alpha_a)}$$

codeword	received vector	
	soft information	hard information
(-1,-1,-1)	(-0.942,-0.736,0.155)	(-1,-1,1)
(-1,-1,1)	(0.034,-0.941,1.569)	(1,-1,1)
(1,1,-1)	(0.815,0.611,-1.837)	(1,1,-1)
(1,1,1)	(0.241,1.571,1.681)	(1,1,1)
(-1,-1,1)	(-2.282,-1.92,2.231)	(-1,-1,1)
(1,1,1)	(1.964,-0.043,0.039)	(1,-1,1)
(-1,-1,-1)	(0.109,-1.28,-1.65)	(1,-1,-1)

Table 4.4: Transformation from soft to hard information

- values $d(r_i, \alpha_j)$, where $1 \leq j \leq q$ are values of column for given element r_i.

Example 4.23.
Given is vector of information elements:

$$m = (\alpha, \alpha^2, 1)$$

Codeword for $RS(7,3)$ code created by original method is as follows:

$$c = (\alpha^5, \alpha^6, \alpha, 0, \alpha^6, 0, \alpha^5)$$

Codeword can be also expressed in vector form:

$$c = ((1,1,1),(1,1,0),(0,0,1),(0,0,0),(1,1,0),(0,0,0),(1,1,1))$$

After modulation $0 \to 1$ and $1 \to -1$, codeword is as follows:

$$c = ((-1,-1,-1),(-1,-1,1),(1,1,-1),(1,1,1),(-1,-1,1),(1,1,1),(-1,-1,-1))$$

Let the codeword be disrupted by the method described previously for some parameter of noise frequency and noise amplitude. In table 4.4 there are shown elements of codeword and elements of disrupted word. Hard information is determined from soft information, that elements which are less or equal zero are transformed to -1, and elements greater than zero are transformed to 1. If received vector would be decoded as hard vector, then there would occur 4 errors. Kötter-Vardy algorithm can help to transform soft information to hard information in such way, that it would be possible to correct more errors. Using received vector, there is created reliability matrix. For next elements of received vector there is computed probability, that given element is some element in $GF(q)$. Reliability matrix fo received vector and parameter $s = 20,5$ is as follows:

$$\Pi = \begin{bmatrix} 0 & 0 & 0 & 0.99 & 0 & 0.22 & 0 \\ 0 & 0 & 0 & 0.012 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.13 & 0 \\ 0 & 0.7 & 0 & 0 & 0 & 0.41 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.24 & 0.91 \\ 0.0079 & 0 & 0 & 0 & 0 & 0 & 0.09 \\ 0.99 & 0.3 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

For clarity, values less than 0.0001 are denoted as 0. Furthermore, sometimes sum of elements is close to 1 and to equal 1, because of floating point number rounding. In the last but one column it can be observed the biggest uncertainty. Received element is $(1.964, -0.043, 0.039)$. The greatest probability is, that this element is α^2, which is denoted as $(0, 1, 0)$. It would be recognized so as hard information. But probability is also big, that given element can be also 0 – $(0, 0, 0)$, α – $(0, 0, 1)$ and α^4 – $(0, 1, 1)$. First position is most firm, which is 1.964, and it can be described as "strong" 1. Values -0.043 and 0.039 are more or less between -1 and 1, so probability that they are 1 or 0 is almost equal.

The next step is transformation of reliability matrix Π to multiplicity matrix M. Kötter-Vardy algorithm is used to this task. Returned matrix from Kötter-Vardy algorithm for parameter $s = 35$ is as follows:

$$M = \begin{bmatrix} 0 & 0 & 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 2 & 0 \\ 0 & 3 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 4 & 3 & 0 & 0 & 4 & 0 & 0 \end{bmatrix}$$

Points, which will be used to interpolate polynomial $Q(x, y)$ are (with multiplicities of zero):
$(1, \alpha^6)$ $m_1 = 4$
(α, α^6) $m_2 = 3$
(α, α^2) $m_3 = 3$
(α^2, α) $m_4 = 4$
$(\alpha^3, 0)$ $m_5 = 3$

(α^4, α^6) $m_6 = 4$
(α^5, α^4) $m_7 = 2$
(α^5, α^2) $m_8 = 3$
(α^5, α) $m_9 = 2$
$(\alpha^5, 0)$ $m_{10} = 2$
(α^6, α^5) $m_{11} = 2$
(α^6, α^4) $m_{12} = 3$

Comparing Kötter-Vardy construction of set of points to interpolate polynomial $Q(x, y)$ and original method for Guruswami-Sudan algorithm, it can be observed that:

- number of points used to interpolation for $RS(n, k)$ code can vary than n,
- the value of order of multiplicity of zeroes can vary from point to point.

Number of all iterations which must be done to interpolate polynomial $Q(x, y)$ is as follows:

$$C = \frac{1}{2}\sum_{i=1} m_i(m_i + 1)$$

Maximal number of returned polynomials L during factorization of $Q(x, y)$ polynomial can be expressed as:

$$L \geq \frac{\Delta_{1,k-1}(C)}{k-1}$$

$$\Delta_{w_X,w_Y}(v) = \min\{\delta \in \mathbb{Z} : N_{w_X,w_Y}(\delta) > v\}$$
$$N_{w_X,w_Y}(\delta) = |\{x^i y^j : i, j \geq 0 \text{ i } iw_X + jw_Y \leq \delta\}|$$

Example 4.24.
For previous set of points and code $RS(7, 3)$, weighted degree is $(1, 2)$ w-revlex. Number of iterations is as follows:

$$C = \frac{1}{2}\sum_{i=1} m_i(m_i + 1) = 72$$

Minimal value of value of weighted degree δ, which satisfies that number of monomials of type $x^i y^j$ is greater than 72 is as follows:

$$\Delta_{1,2}(72) = \min\{\delta \in \mathbb{Z} : N_{1,2}(\delta) > 72\}$$

Monomial x^{16}, where $\deg_{1,2} = 16$, is on 73th place on monomial list of order $(1, 2)$ w-revlex, so:

$$\Delta_{1,2}(72) = 16$$

Maximal number of returned polynomials is at least:

$$L \geq \frac{16}{3-1} = 8$$

If there are given points to interpolate polynomial $Q(x, y)$, their orders of multiplicity of zeroes, maximal number of returned polynomials, then interpolation of $Q(x, y)$ can be started.
Modified Kötter algorithm for Kötter-Vardy algorithm is as follows:

1. Given is following input data:
   - maximal number of returned polynomials L,
   - set of h points (α_i, β_i) for $i = \{1, 2, \ldots, h\}$, created by Kötter-Vardy algorithm,
   - set of multiplicities of zeroes for each point m_i for $i = \{1, 2, \ldots, h\}$,
   - order $(1, k-1)$ w-revlex for $RS(n, k)$ code,
   - set of ordered tables of pairs (r, s) for each point (α_i, β_i) – K_i for $i = \{1, 2, \ldots, h\}$.
2. Create $L + 1$ polynomials $g_j(x, y) = y^j$, where $j = \{0, 1, \ldots, L\}$.
3. Set variable $i = 1$, which will point number of current evaluated point (α_i, β_i).
4. If $i > h$, then go to point 14, else set point $(a, b) = (\alpha_i, \beta_i)$ and continue.
5. Set variable $j = 1$, which will point pair (r, s) in table K_i.
6. If $j > \binom{m_i+1}{2}$, the increase variable $i = i + 1$ and return to point 4, else set as current pair $(r, s) = K_i[j]$.
7. For each polynomial $g_j(x, y)$, where $j = \{0, 1, \ldots, L\}$, compute value of discrepancy Δ_j, which is used to modify polynomial $g_j(x, y)$:

$$\Delta_j = Q_{r,s} g_j(a, b)$$

8. Create set J, which consists of indices j, where $\Delta_j \neq 0$.
9. If set J is empty, then increase variable $j = j + 1$ and go to point 6, else continue.
10. In set J find polynomial of least weighted degree and remember its index j^*:

$$j^* = \arg\min\{g_j(x, y) : j \in J\}$$

11. Set two auxiliary variables:

$$f = g_{j^*}(x, y) \qquad \Delta = \Delta_{j^*}$$

12. For each polynomial $g_j(x, y)$, where $j \in J$, do:

- if $j \neq j^*$, compute:

$$g_j(x, y) = \Delta g_j(x, y) - \Delta_j f$$

- if $j = j^*$, compute:

$$g_j(x, y) = \Delta(x - a)f$$

13. Increase variable $j = j + 1$ and go to point 6.
14. From set of polynomials $g_j(x, y)$, choose this one of least weighted degree and return it:
$$Q(x, y) = g_j(x, y),$$
where $\deg_w g_j(x, y)$ is least for $j = \{0, 1, \ldots, L\}$.

Original Roth-Ruckenstein algorithm can be used to factorize polynomial $Q(x, y)$.

5. Implementation

Algorithms described previously were implemented in hardware and in software. Following algorithms were implemented in software:

- cyclic decoding with use of remainder of division of received vector by generator polynomial,
- Peterson-Gorenstein-Zierler algorithm,
- Berlekamp-Massey algorithm,
- Sugiyama algorithm without erasures,
- Sugiyama algorithm with erasures,
- Forney algorithm and Chien search,
- Guruswami-Sudan algorithm, and Kötter algorithm and Roth-Ruckenstein algorithm for hard decoding,
- Guruswami-Sudan algorithm, and Kötter-Vardy algorithm and modified Kötter algorithm for soft decoding.

Algorithms which were implemented in hardware:

- error trapping decoder,
- Peterson-Gorenstein-Zierler algorithm,
- Berlekamp-Massey algorithm,
- Sugiyama algorithm,
- Forney algorithm, Chien search.

Algorithms are based on mathematical functions of Galois fields, and there were also implemented functions for polynomials of one and two variables, vectors and matrices.

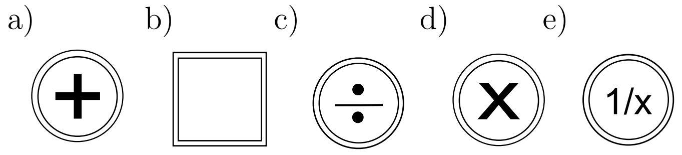


Figure 5.1: Symbols of elements: a) element which adds two elements of field or two polynomials, b) registers which holds one element of field, c) element which divides two elements of field or two polynomials, d) element which multiplies two elements of field or two polynomials, e) element which gives out multiplicative inverse of element in field.

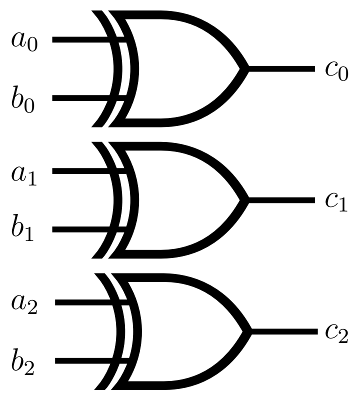


Figure 5.2: Adder for $GF(8)$.

5.1 Arithmetic operations for Galois fields

Implementations of operations in Galois fields in hardware and software are different. It is mainly caused by representation of field elements. In software Galois field elements are represented as positive integer numbers $0, 1, 2, \ldots, q-1$, and in hardware as binary vectors. There are used Galois fields of characteristic 2. Characteristic of fields doesn't complicate implementation in software much, but it affects much hardware implementation. In software implementation biggest field which can be used is $GF(256)$, because elements are represented as bytes. Of course, elements can be represented by bigger data types than byte, but there was no need to use so big fields. Elements of $GF(256)$ use all 8 bits of byte, and elements of fields less than $GF(256)$ waste some space. In software implementation there can be used fields from $GF(4)$ to $GF(256)$, and for hardware implementation it is more static, because decoders are implemented just for one field.

In software implementation there are two solutions for multiplication of Galois field elements: LUT tables and computing multiplication in real time with use of Zech logarithm. LUT tables don't require implementation of arithmetic operations, and result of multiplication is read straight from memory addressed by multiplication operands. The drawback is that data must be stored in memory. With increase of number of field elements, multiplication will work slower because data in cache memory is exchanged more often. However for field $GF(256)$ number of elements which must be stored is (results of multiplication):

$$N_{LUT} = 2^{2m-1} - 3 \cdot 2^{m-1} + 1$$

$$N_{LUT} = 2^{15} - 3 \cdot 2^{7} + 1 = 32768 - 384 + 1 = 32385$$

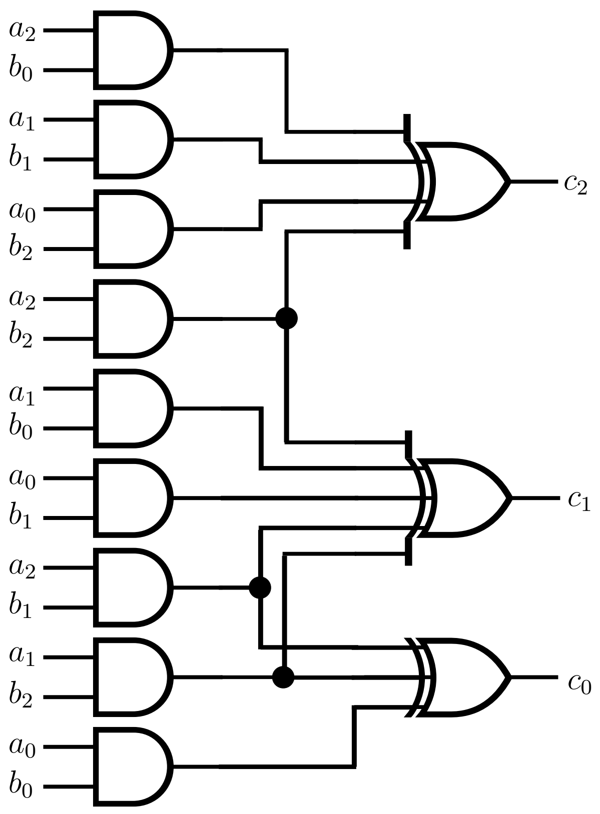


Figure 5.3: Multiplication element for $GF(8)$.

To store results of multiplication and addition there is needed $2 \cdot 32385 = 64770$ bytes of memory, which is not so big constraint for current technology.

In the case of computing result in real time, there must be implemented units which multiply and add elements of Galois field. Furthermore, there is needed table of Zech logarithms values. Zech logarithms can be computed with use of Imamura algorithm. For field $GF(q)$ there must be stored q Zech logarithms. For example, for $GF(256)$ there must be stored 256 bytes. Addition in real time complicates structure of decoder, but it doesn't require much of memory, so it can be used in systems, where memory is limited.

For fields of characteristic 2 subtraction is equivalent to addition, so all solutions of addition can be used for subtraction.

Division can be processed as multiplication of dividend and multiplicative inverse of divisor. Multiplicative inverse of element x in Galois field $GF(q)$ is $q - x$ except for 1 it is 1. Then multiplication is done.

In hardware implementation elements of field are stored in registers. Registers consist of flip flops, where one flip fop stores one bit of element vector.

For $GF(2^m)$ registers is of length m. There is used representation as polynomials for Galois field elements. Its advantage is that it's easy to construct multiplication units for this representation. Addition is done with use of XOR function. Exemplary adder of two elements in $GF(8)$ is shown on picture 5.2. Each bit of the first operand is XOR'ed with adequate bit of secoind operand. Multiplication can be done as multiplication of two polynomials. Let one of factor of product be $a(x) = a_{m-1}x^{m-1} + \ldots + a_1x + a_0$, and the other is $b(x) = b_{m-1}x^{m-1} + \ldots + b_1x + b_0$. Operations are done in $GF(2^m)$ created by primitive polynomial $p(x) = p_mx^m + \ldots + p_1x + p_0$. Multiplication can be expressed as follows:

$$c(x) = a(x)b(x) \bmod p(x)$$

Example 5.1.
There will be designed multiplication unit for elements of field $GF(8)$ created by primitive polynomial $p(x) = x^3 + x + 1$. Elements of $GF(8)$ can be represented as polynomials of type:

$$a_2x^2 + a_1x + a_0$$

Multiplication can be expressed as:

$$c(x) = a(x)b(x) \bmod p(x)$$

$$c(x) = (a_2x^2 + a_1x + a_0)(b_2x^2 + b_1x + b_0) \bmod (x^3 + x + 1)$$

$$c(x) = (a_2b_0+a_1b_1+a_0b_2+a_2b_2)x^2+(a_1b_0+a_0b_1+a_2b_2+a_2b_1+a_1b_2)x+(a_0b_0+a_2b_1+a_1b_2)$$

Addition can be proceeded by XOR function, and multiplication can be done with AND function. Multiplication unit for any two elements of field is shown on picture 5.3.

Sometimes there is need to multiply elements of Galois field by constant. Unit which multiplies elements by constant is less complicated than unit which multiplies any elements of field. To construct unit which multiplies by constant, there must be known vector representation of constant and m linear independent vectors for $GF(2^m)$. There are used to construct multiplication unit by constant α^c linear indenpendent vectors for elements $1, \alpha, \ldots, \alpha^{m-1}$.

Taking into account table 5.1 there can be devised functions which will be used to create multiplication unit by constant:

$$a(x)R^{(\alpha^c)}(x) = b(x)$$

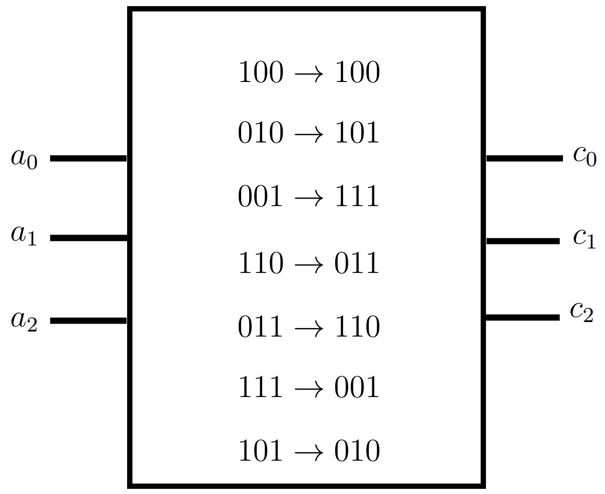


Figure 5.4: Unit which computes multiplicative inverses in $GF(8)$.

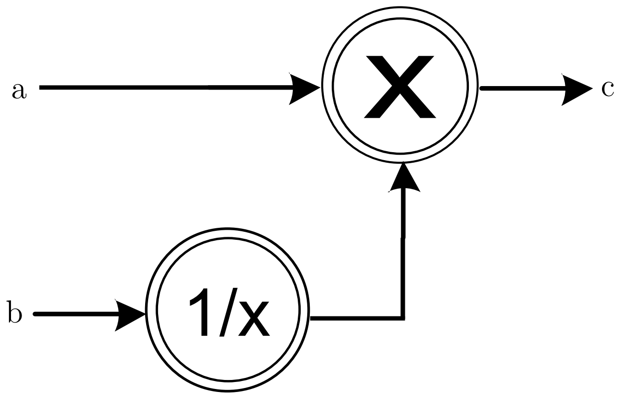


Figure 5.5: Division of element a by element b in field.

α^i	a_0	a_1	$\dots$	a_{m-1}	α^{c+i}	b_0	b_1	$\dots$	b_{m-1}
1	1	0	$\dots$	0	α^c	$b_0^{(1)}$	$b_1^{(1)}$	$\dots$	$b_{m-1}^{(1)}$
α	0	1	$\dots$	0	α^{c+1}	$b_0^{(\alpha)}$	$b_1^{(\alpha)}$	$\dots$	$b_{m-1}^{(\alpha)}$
$\vdots$					$\vdots$				
α^{m-2}	0	0	$\dots$	1	α^{c+m-2}	$b_0^{(\alpha^{m-2})}$	$b_1^{(\alpha^{m-2})}$	$\dots$	$b_{m-1}^{(\alpha^{m-2})}$

Table 5.1: Multiplication of linear independent vectors by constant.

α^i	a_0	a_1	a_2	α^{4+i}	b_0	b_1	b_2
1	1	0	0	α^4	0	1	1
α	0	1	0	α^5	1	1	1
α^2	0	0	1	α^6	1	0	1

Table 5.2: Multiplication of vectors by constant α^4.

$$\begin{aligned}
b_0 &= b_0^{(1)}a_0 + b_0^{(\alpha)}a_1 + \ldots + b_0^{(\alpha^{m-2})}a_{m-1}\\
b_1 &= b_1^{(1)}a_0 + b_1^{(\alpha)}a_1 + \ldots + b_1^{(\alpha^{m-2})}a_{m-1}\\
&\vdots\\
b_{m-1} &= b_{m-1}^{(1)}a_0 + b_{m-1}^{(\alpha)}a_1 + \ldots + b_{m-1}^{(\alpha^{m-2})}a_{m-1}
\end{aligned}$$

Using these functions, there can be created multiplication unit by constant.

Example 5.2.
There will be designed multiplication unit by constant α^4 for field $GF(8)$. Element α^4 in polynomial representation is as follows:

$$R^{(\alpha^4)}(x) = x^2 + x$$

There are needed 3 linear independent vectors (here they are represented as polynomials):

$$R^{(1)}(x) = 1$$
$$R^{(\alpha)}(x) = x$$
$$R^{(\alpha^2)}(x) = x^2$$

Using table 5.2, there can be devised following equations:

$$\begin{aligned}
b_0 &= 0 \cdot a_0 + 1 \cdot a_1 + 1 \cdot a_2\\
b_1 &= 1 \cdot a_0 + 1 \cdot a_1 + 0 \cdot a_2\\
b_2 &= 1 \cdot a_0 + 1 \cdot a_1 + 1 \cdot a_2
\end{aligned}$$

After elimination of not needed values:

$$\begin{aligned}
b_0 &= a_1 + a_2\\
b_1 &= a_0 + a_1\\
b_2 &= a_0 + a_1 + a_2
\end{aligned}$$

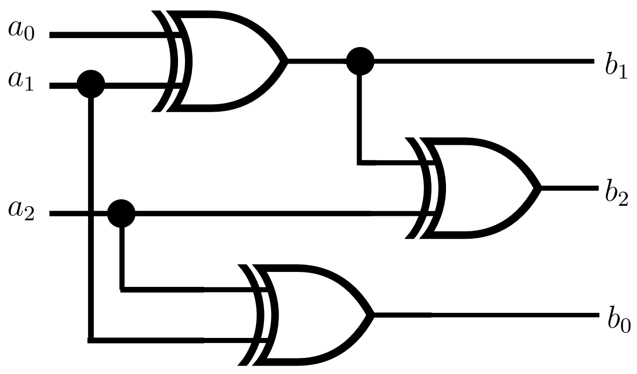


Figure 5.6: Multiplication unit by constant α^4 in $GF(8)$.

Multiplication unit by constant α^4 is shown on picture 5.6.

Computing multiplicative inverse of some element can be worked out as simple memory, which stores pairs of Galois field element and its multiplicative inverse. Exemplary unit which computes multiplicative inverse is shown on picture 5.4. There can be made reduction of stores pairs by half, because elements are first half elements is symmetric to the second half. Input element will be compared with two elements in pair and there will be returned element which is not equal to input element.

Division can be done as multiplication of multiplicative inverse of divisor. Division for Galois field elements is shown on picture 5.5.

5.2 Polynomials

Polynomials, whose coefficients are Galois fields elements, are broadly used in Reed-Solomon decoders. The often used operations associated with polynomials are:

- addition of polynomials,
- multiplication of polynomials,
- division of polynomials (Sugiyama algorithm),
- computing remainder of polynomial division (cyclic decoding),
- evaluation of polynomials.

Addition of polynomials can be done by addition of coefficients on the same position in both polynomials.

Multiplication of polynomials gives as a result polynomial of degree of sum of degrees of multiplied polynomials. On picture 5.7 there is shown unit for multiplication of polynomials. One of the multiplication factors is polyno-

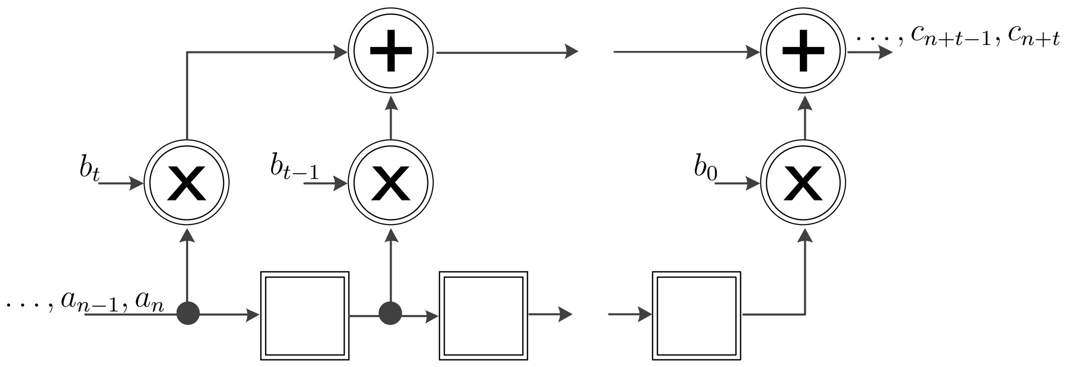


Figure 5.7: Multiplication of polynomial $a(x)$ by polynomial $b(x)$. The result is polynomial $c(x)$.

mial $b(x)$ of degree t. On this position there can be placed any polynomial of degree not greater than t. If there is need that factor on this place is polynomial of degree m, where $m < t$, then on position b_0 must be placed coefficient of this polynomial on position x^0 and on next positions there is placed rest of coefficients. Positions from t to $t-m+1$ are filled with zeroes. All registers must be filled with zeroes before polynomial multiplication. Combinational unit for polynomial multiplication can be designed similar to multiplication of two Galois field elements represented as polynomials.

Division of polynomials is base of Euclidean algorithm which is used in Sugiyama algorithm. On picture 5.8 is shown unit for polynomial division. This unit is used also to compute remainder of polynomial division. Maximal degree of divisor is t. Any polynomial of degree not greater than t can be divisor. If divisor is of degree m, where $m < t$, then highest coefficient of polynomial is on position b_t, and on lower positions is stored remainder of coefficients. Positions from $t-m-1$ to 0 are filled with zeroes. All registers must be filled with zeroes before division.

Evaluation of polynomial can be done by multiplication of coefficients with consecutive powers of variable and after that sum of all products is done. Number of multiplications can be reduced with use of Horner's rule. Given is polynomial:

$$w(x) = a_t x^t + \ldots + a_1 x + a_0$$

This polynomial can be also rewritten as:

$$w(x) = x(x(x(\ldots x(a_t x + a_{t-1}) + a_{t-2}) \ldots) + a_0$$

On picture 5.10 is shown unit, which sequentially evaluates polynomial, and on picture 5.9 is shown unit, which evaluates polynomial combinationally.

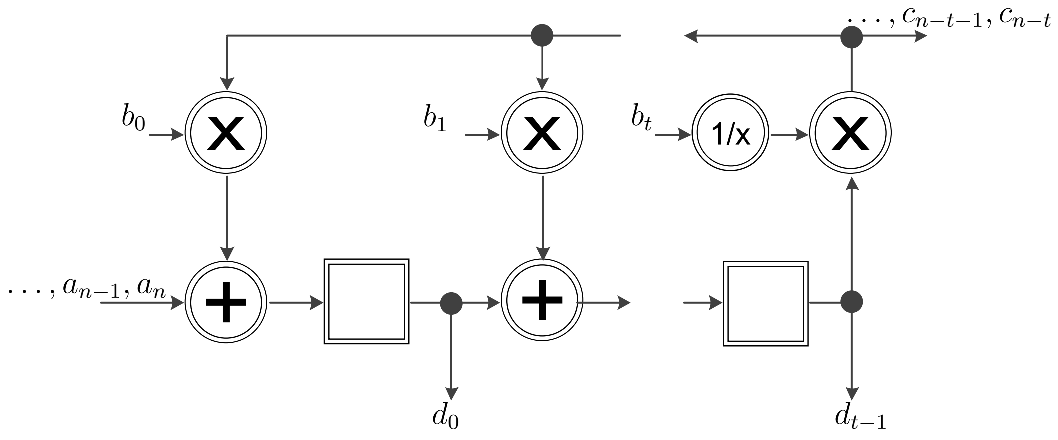


Figure 5.8: Division of polynomial $a(x)$ by polynomial $b(x)$. Result of division is polynomial $c(x)$ and remainder is polynomial $d(x)$.

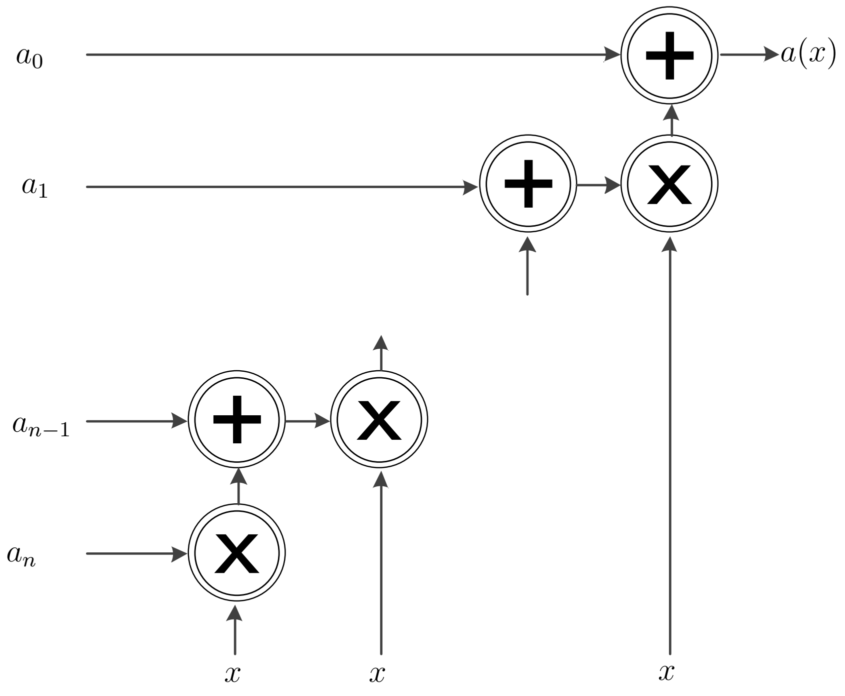


Figure 5.9: Combinational evaluation of polynomial with use of Horner's rule.

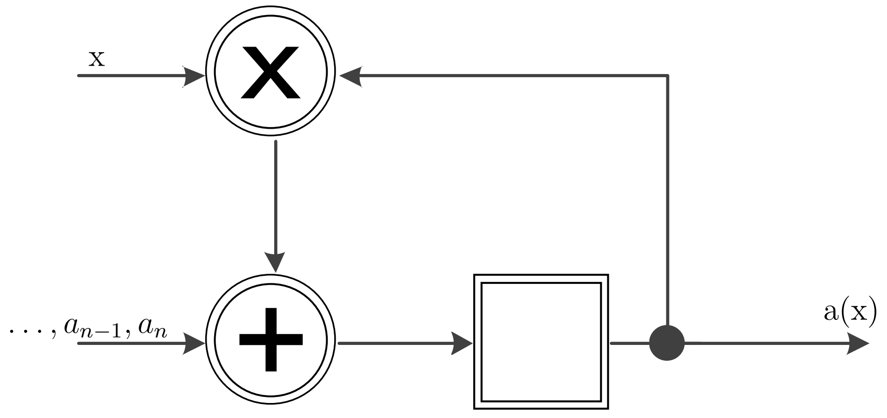


Figure 5.10: Sequential evaluation of polynomial with use of Horner's rule.

Sequential unit's registers must be filled with zeroes before polynomial evaluation.

5.3 Error trapping decoder

Error trapping decoder decodes received vectors cyclically [9]. It is used for systematic $RS(n,k)$ code with error correction capability of t. It consists of four parts:

- syndrome register,
- buffer register,
- unit which checks Hamming weight of syndrome,
- control unit.

Syndrome registers is built on division unit which divides received vector by $g(x)$. It consists of $2t$ registers which store Galois field elements, not more than $2t$ multiplication units and $2t$ addition units. There are processed two operations, division of received vector by code generator, and also there is computed remainder of this division. Remainder of this division is syndrome vector.

Buffer register consists of k registers which store Galois field elements.

Unit which checks Hamming weight of syndrome is used to signal situation when Hamming weight of syndrome is less or equal to error correction capability t. It can be constructed as counter of ones and comparator. If given element of Galois field represented as vector is nonzero vector, then it can be interpreted as element 1. To this transformation OR gate can be used.

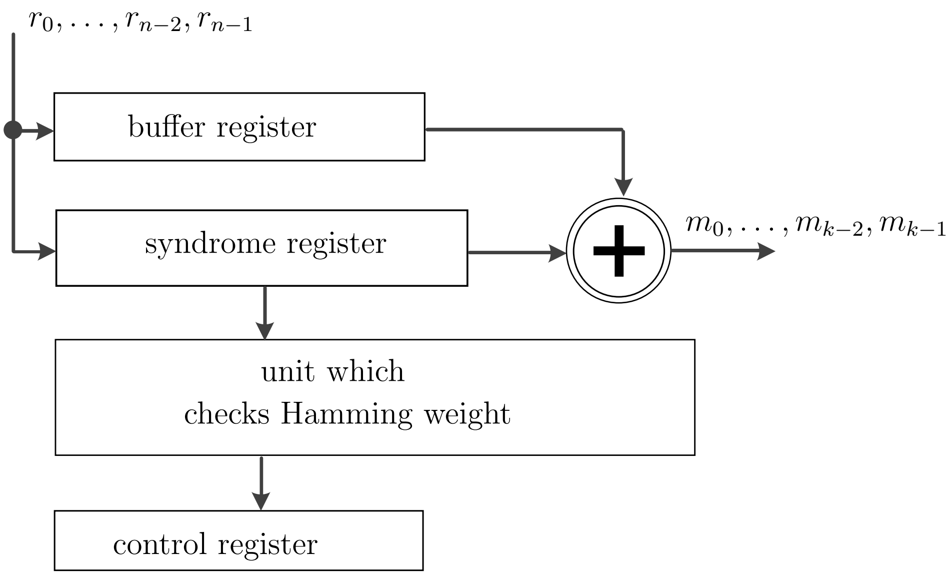

$r_0, \dots, r_{n-2}, r_{n-1}$
buffer register
syndrome register
$m_0, \dots, m_{k-2}, m_{k-1}$
unit which checks Hamming weight
control register

Figure 5.11: Error trapping decoder - general overview.

Next ones are counted and number of ones is compared with error correction capability of code.

Control unit is used to control buffers b_1 and b_2, and syndrome registers and buffer register. Control unit's work is influenced by signals received from unit which checks Hamming weight of syndrome register.

Error trapping decoding algorithm is as follows:

1. Fill all registers with zeroes.
2. Fill buffer register and syndrome register in k cycles with information elements of received vector. Freeze state of buffer register. Gate b_1 is open, and gate b_2 is closed.
3. In the next $n - k$ cycles feed syndrome register with remainder of received vector.
4. Check Hamming weight of syndrome.
    - if Hamming weight of syndrome is less or equal to error correction capability, then it means, that there are no errors or there are contained within control elements of received vector. Open gate b_2 and close gate b_1. In next k cycles move out simultaneously elements from buffer register and syndrome register. Correct errors if it is necessary.
    - if Hamming weight of syndrome is greater than error correction capability, then continue.
5. Shift syndrome register in $1 \leq i \leq k$ cycles.

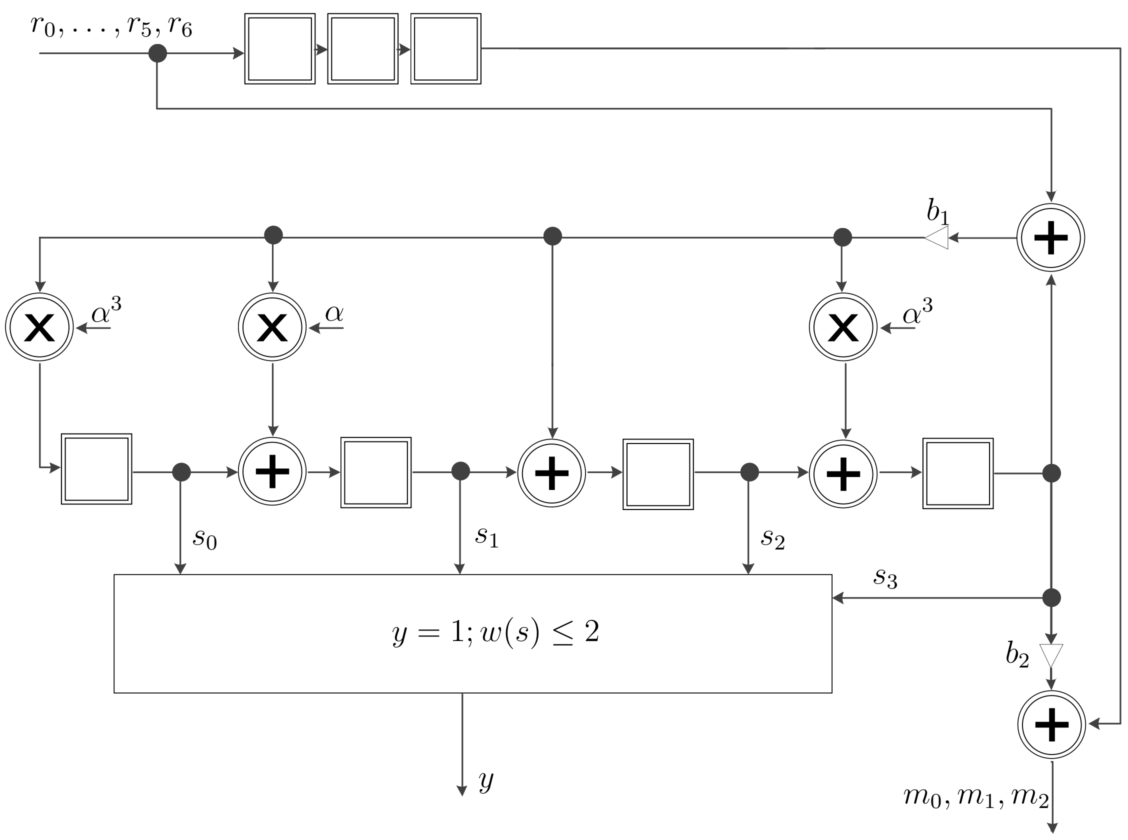


Figure 5.12: Error trapping decoder for $RS(7, 3)$ code with code generator $g(x) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$.

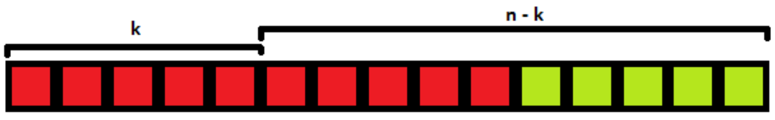


Figure 5.13: Errors which are corrected during step 4 for $RS(15, 5)$ code. With the red color there are marked positions, where error can occur, and with green color are marked positions where error cannot occur.

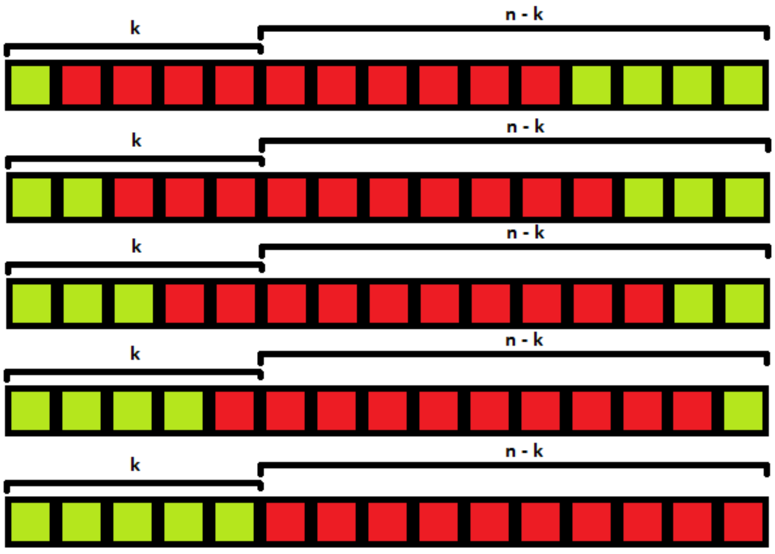


Figure 5.14: Errors which are corrected during step 5 for $RS(15, 5)$ code. With the red color there are marked positions, where error can occur, and with green color are marked positions where error cannot occur.

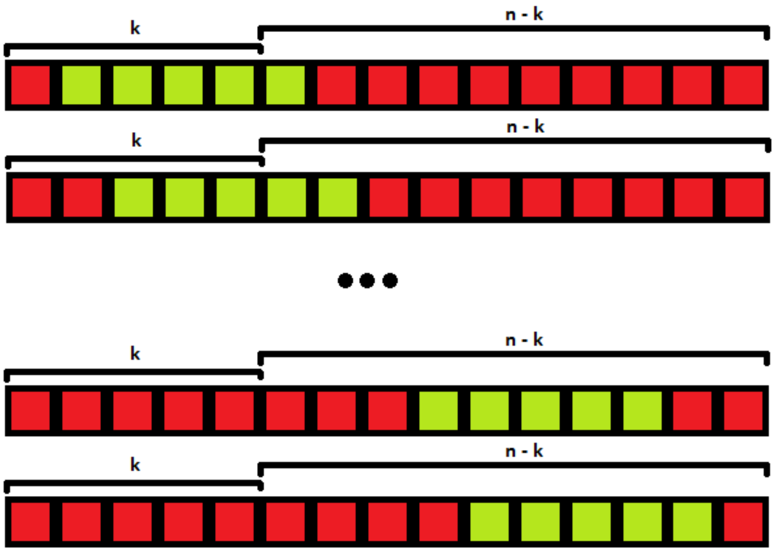


Figure 5.15: Errors which are corrected during step 6 for $RS(15, 5)$ code. With the red color there are marked positions, where error can occur, and with green color are marked positions where error cannot occur.

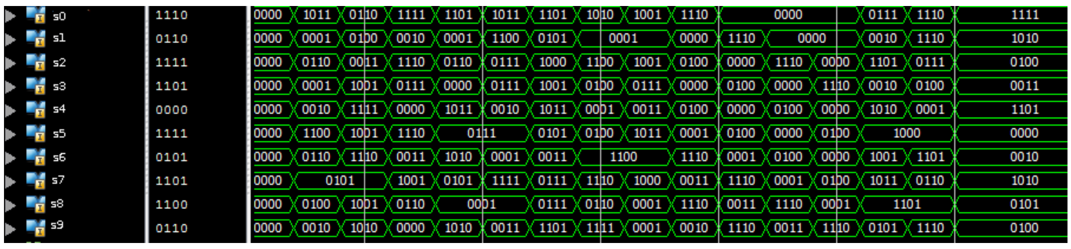


Figure 5.16: Error trapping decoding for $RS(15, 5)$. Syndrome register's state during first $n = 15$ cycles.

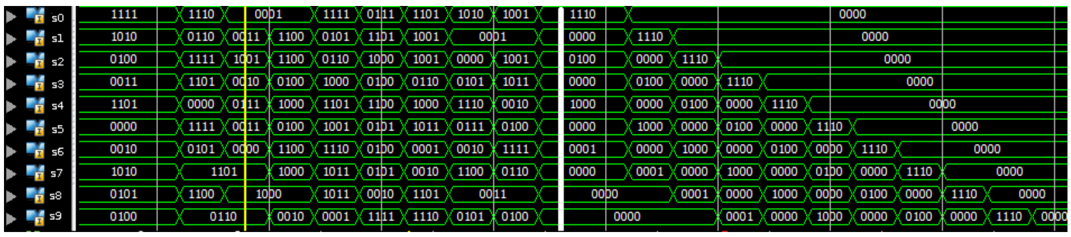
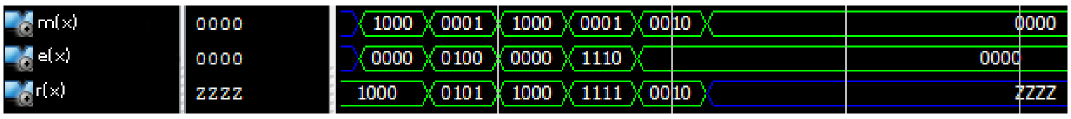


Figure 5.17: Shifting syndrome register until Hamming weight is less than error correction capability of code (pointed by red line).

- if during i-th cycle Hamming weight of syndrome is less or equal to error correction capability of code, then errors are contained on $r_{i-1}, \dots, r_{n-k+i-1}$ positions of received vector. Gate b_1 and b_2 are closed, syndrome register is frozen. From buffer register there are moved out first i elements, then gate b_2 is opened and syndrome register is shifted out and eventually errors are corrected.
- if during $1 \leq i \leq k$ cycles Hamming weight was always higher than error correction capability, then continue.

6. Shift syndrome register $1 \leq j \leq n - k$ times.
   - if during j-th cycle Hamming weight of syndrome is less or equal to error correction capability of code, then errors are contained on $r_0, \dots, r_{j-1}$ and on $r_{n-j-1}, \dots, r_{n-1}$ positions. Gate b_1 is closed and syndrome register is shifted $n - k - j$ times. After that gate b_2 is opened and with next cycles content of syndrome register and buffer register are moved out.
7. If after n cycles there was found no syndrome with Hamming weight less or equal to error correction capability, then signal that there occured non correctable error pattern.


m(x) 0000 1000 0001 1000 0001 0010 0000
e(x) 0000 0000 0100 0000 1110 0000
r(x) zzzz 1000 0101 1000 1111 0010 ZZZZ
Figure 5.18: Correction of received vector $r(x)$ with syndrome register $e(x)$.

Example 5.3.
Given is code $RS(15,5)$ and following codeword:

$$c = ((1000),(0001),(1000),(0001),(0010),(1100),(0011),(1010),\\ (0101),(0010),(1110),(0011),(1100),(0001),(0100))$$

Error vector is as follows:

$$e = ((0000),(0100),(0000),(1110),(0000),(0000),(0000),(0000),\\ (0000),(0000),(0000),(0000),(0011),(0000),(1000))$$

Received vector is as follows:

$$r = ((1000),(0101),(1000),(1111),(0010),(1100),(0011),(1010),\\ (0101),(0010),(1110),(0011),(1111),(0001),(1100))$$

To buffer register are moved in following elements $r_0,\dots,r_4$:

$$(1000),(0101),(1000),(1111),(0010)$$

Entire received vector is shifted to syndrome register, what is is shown on picture 5.16. There is checked Hamming weight of syndrome. It is greater than error correction capability of code $t = 5$, so syndrome register is cyclically shifted. First $k = 5$ shifts didn't result with appropriate syndrome. Then syndrome is shifted $j = 4$ times more, what is shown on picture 5.17. There is found syndrome with Hamming weight less than error correction capability. Syndrome must be then shifted $n - k - j$ times, which is 6 times and then received vector is corrected, what is shown on picture 5.18.

Cyclic decoding with multithreading

Multithreading can be used to boost speed of cyclic decoding which is finding remainder of division of received vector by code generator polynomial. This remainder should be of Hamming weight less or equal to error correction capability. Division of polynomials is not dependable from previous division, so this task can be splitted within threads. For example one thread can decode original received vector and second vector can decode received vector shifted $\frac{n}{2}$ times. First thread which finds solution, signals it, corrects received vector and returns it.

5.4 Peterson-Gorenstein-Zierler algorithm

Decoding with Peterson-Gorenstein-Zierler algorithms consists of three main steps:

- finding number of errors,
- finding positions of errors,
- finding values of errors.

Finding number of errors can be done by computation of determinant of matrix $M^{(v)}$ (4.8) for $v = t$ and then by checking whether matrix is invertible or not. If matrix isn't invertible then there is computed determinant for matrix $M^{(v)}$, where new $v = t-1$ etc. If matrix is invertible, then there must be solved set of equations 4.8. The result is polynomial, whose multiplicative inverses of roots are positions of errors. Syndromes values and error positions are used to compute error values.

As it was stated earlier, determinant of matrix can be computed with Laplace expansion. Laplace expansion is of complexity O(n!). To solve set of equations Gauss elimination is used.

For small values of error correction capability of $RS(n, k)$ code it is easy to design combinational decoder [24].

Example 5.4.
There will be designed combinational decoder for $RS(7, 3)$ code. Error correction capability is $t = 2$. There must be compute $2t = 4$ values of syndromes. Combinational units to evaluation of polynomials will be used 5.9.

Next step is to compute number of errors. If syndromes values are zeroes, then there are no errors or error pattern cannot be recognized. If two errors occured then following set of equations must be solved:

$$\begin{bmatrix} s_1 & s_2 \\ s_2 & s_3 \end{bmatrix} \begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -s_3 \\ -s_4 \end{bmatrix}$$

Above set of equations can be rewritten as:

$$\begin{aligned} s_1\Lambda_2 + s_2\Lambda_1 &= -s_3 \\ s_2\Lambda_2 + s_3\Lambda_1 &= -s_4 \end{aligned}$$

Coefficients of error locator polynomial are:

$$\Lambda_1 = \frac{-s_1 s_4 + s_2 s_3}{-s_2^2 + s_1 s_3}$$

$$\Lambda_2 = \frac{-s_2 s_4 + s_3^2}{s_2^2 - s_1 s_3}$$

It can be observed, that in denominators of coefficients is contained value of determinant of $M^{(2)}$.

If one error occured, then there must be solved following equation:

$$\begin{bmatrix} s_1 \end{bmatrix} \begin{bmatrix} \Lambda_1 \end{bmatrix} = \begin{bmatrix} -s_2 \end{bmatrix}$$

Coefficient of error locator polynomial is:

$$\Lambda_1 = \frac{-s_2}{s_1}$$

Unit on picture 5.19 computes values of coefficients of error locator polynomial $\Lambda(x)$ and computes number of occured errors. Following things can happen according to occured errors:

- if values of syndromes are all zeroes, then there are no errors or error pattern is not recognizable,
- if value $s_2^2 + s_1 s_3$ is not equal to zero, then there occured two errors,
- if value $s_2^2 + s_1 s_3$ is equal to zero and value s_2 is not equal to zero, then one error occured,
- if value $s_2^2 + s_1 s_3$ is equal to zero and value s_2 is equal to zero and at least one of syndromes is not equal to zero, then error pattern cannot be corrected.

Computing zeroes of error locator polynomial can be done by use of combinational units which evaluate polynomial for all elements of Galois field except zero. For $RS(n, k)$ code there are needed n units which compute zeroes.

When positions of errors are known, then there must be found values of errors. There must be solved following set of equations:

$$\begin{aligned} s_1 &= e_{k_1} X_1 + e_{k_2} X_2 + \ldots + e_{k_v} X_v \\ s_2 &= e_{k_1} X_1^2 + e_{k_2} X_2^2 + \ldots + e_{k_v} X_v^2 \\ &\ldots \\ s_v &= e_{k_1} X_1^v + e_{k_2} X_2^v + \ldots + e_{k_v} X_v^v \end{aligned}$$

Error values can be rewritten as: For $v = 1$:

$$e_{k_1} = \frac{s_1}{X_1}$$

$$e_{k_2} = 0$$

For $v = 2$:

$$e_{k_1} = \frac{s_2 + s_1 X_2}{X_1^2 + X_1 X_2}$$

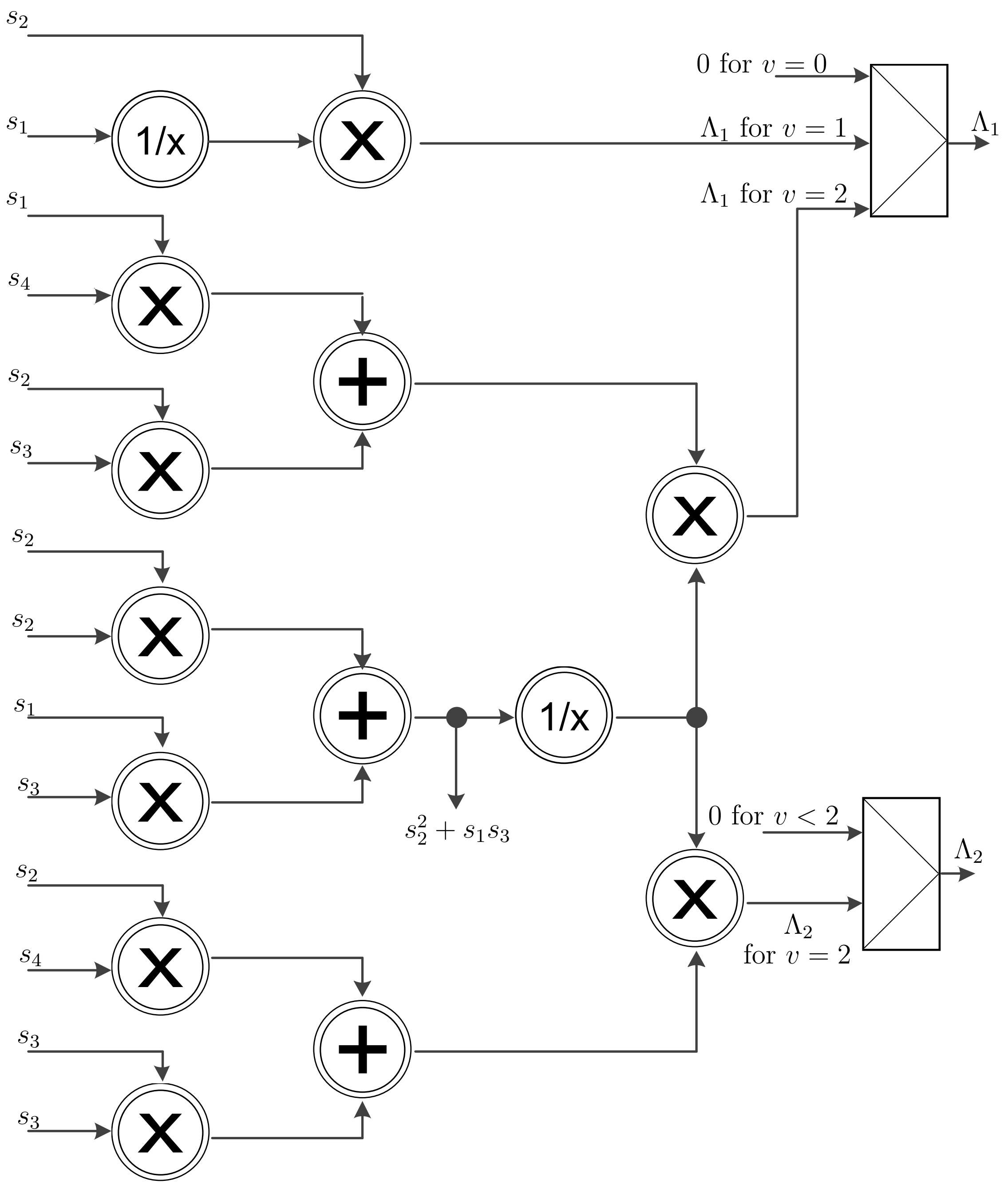


Figure 5.19: Peterson-Gorenstein-Zierler decoder for $RS(n, n-4)$.

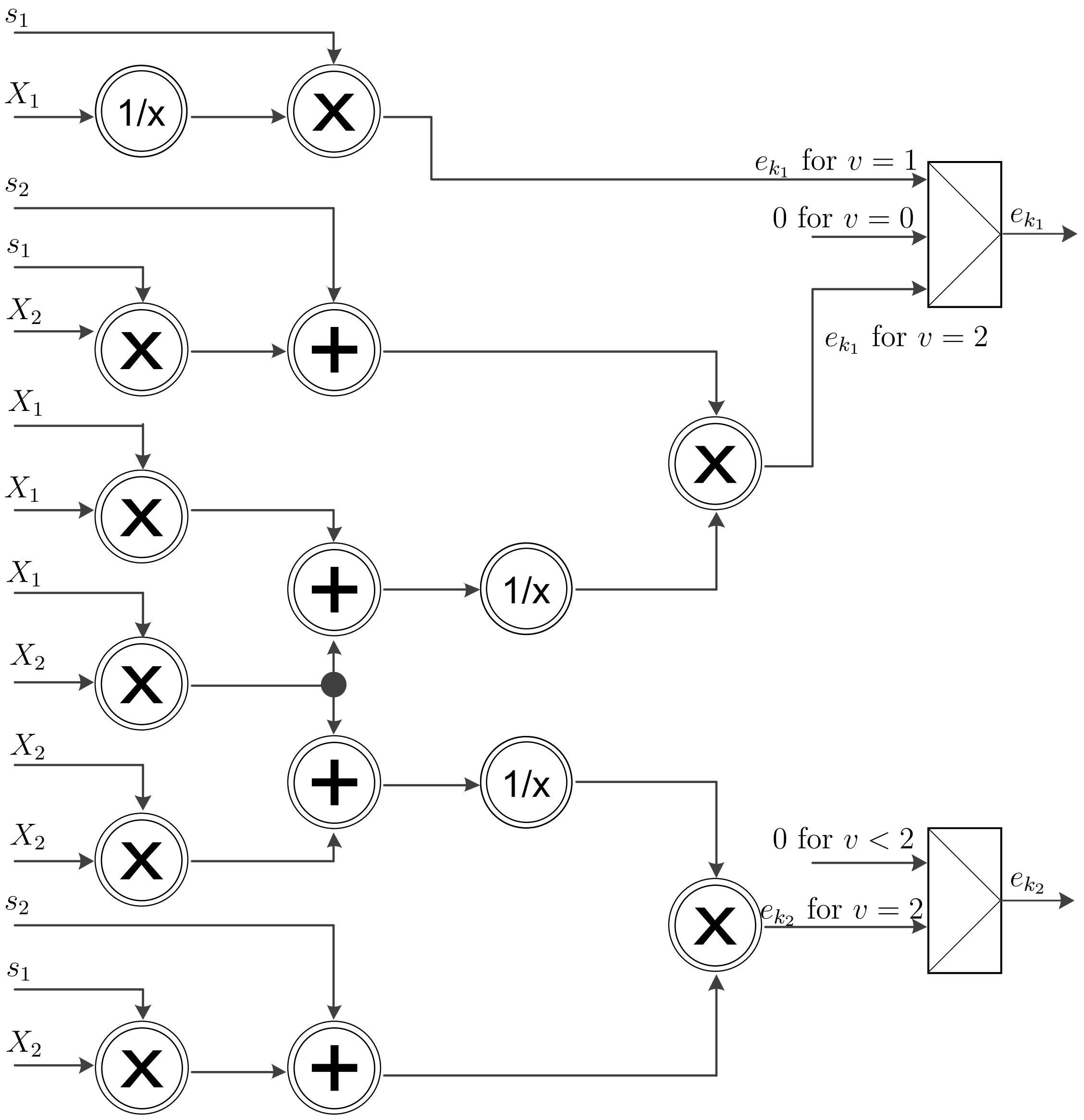


Figure 5.20: Computing error values for $RS(7, 3)$.

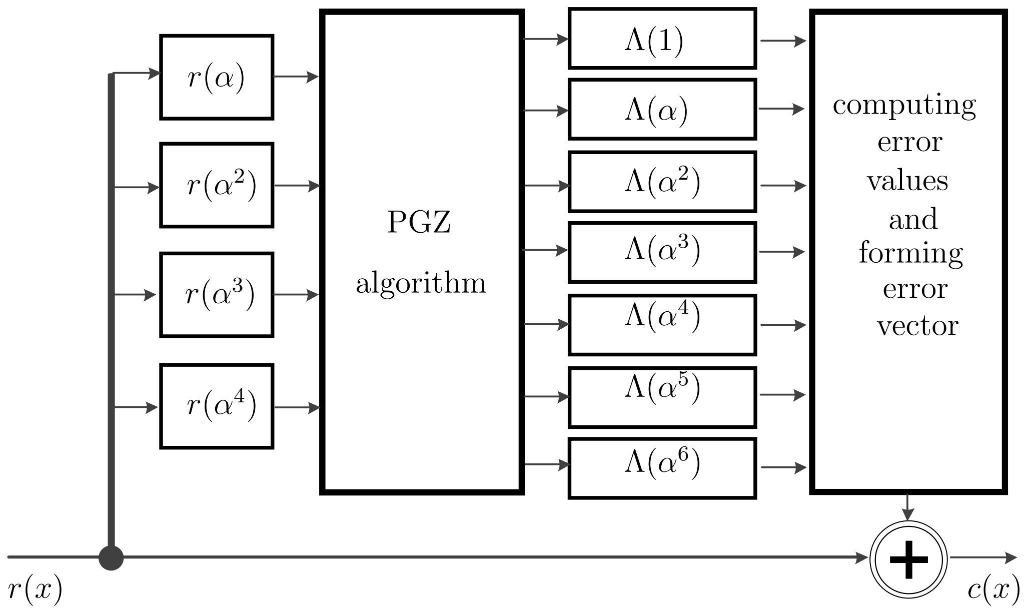


Figure 5.21: Combinational decoder for $RS(7,3)$.

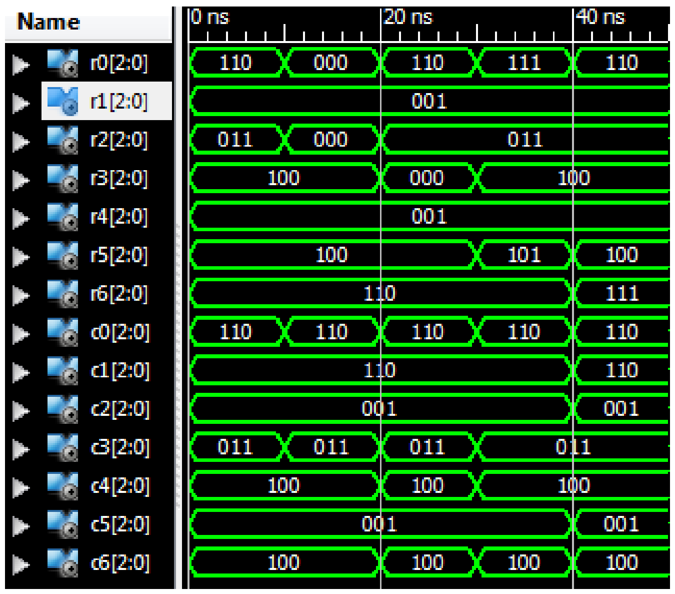


Figure 5.22: Exemplary result of work of decoder for $RS(7,3)$.

$$e_{k_2} = \frac{s_2 + s_1 X_1}{X_1 X_2 + X_2^2}$$

After computing error values, there must be constructed error vector and this vector is then added to received vector in order to correct errors. On picture 5.20 there is shown unit which computes error values. On picture 5.21 there is shown general project of decoder for $RS(7,3)$ code. On picture 5.22 there is shown result of simulation from iSim software. Received vectors are corrected within one cycle.

5.5 Berlekamp-Massey algorithm

Berlekamp-Massey algorithm computes error locator polynomial $\Lambda(x)$ and uses syndromes for that. Schema of decoder for $RS(n, n-4)$ code is shown on picture 5.23. It consists of four registers, one to store value of discrepancy Δ and other 3 registers to store coefficients of error locator polynomial $\Lambda(x)$, auxiliary polynomial $B(x)$ and syndrome vector. First, decoder computes value of discrepancy Δ, multiplying elements of syndrome vector by coefficients of error locator polynomial $\Lambda(x)$. In parallel, there are computed two new values of coefficients of auxiliary polynomial $B(x)$. Multiplexer is used to choose one of these values and it depends on value of discrepancy Δ and result of $2L \leq r - L$. Length of LFSR can be determined by checking current degree of polynomial $\Lambda(x)$. Multiplication by polynomial x can be done by shifting second polynomial one time into higher position. The result of decoder work is error locator polynomial $\Lambda(x)$.

5.6 Sugiyama algorithm

Sugiyama algorithm computes error locator polynomial $\Lambda(x)$ and error evaluator polynomial $\Omega(x)$. It consists of 4 registers which store coefficients of polynomials. There is also included unit for polynomial division. Results of algorithm are multiplied by constant $A_{2,2}(0)$. There is no need however to get rid off this constant, because roots of $\Lambda(x)$ are still the same and in Forney algorithm, these constants are reduced:

$$e_{k_j} = -X_j \frac{A_{2,2}(0)\Omega(X_j^{-1})}{A_{2,2}(0)\Lambda'(X_j^{-1})} = -X_j \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})}$$

On picture 5.24 is shown Sugiyama algorithm.

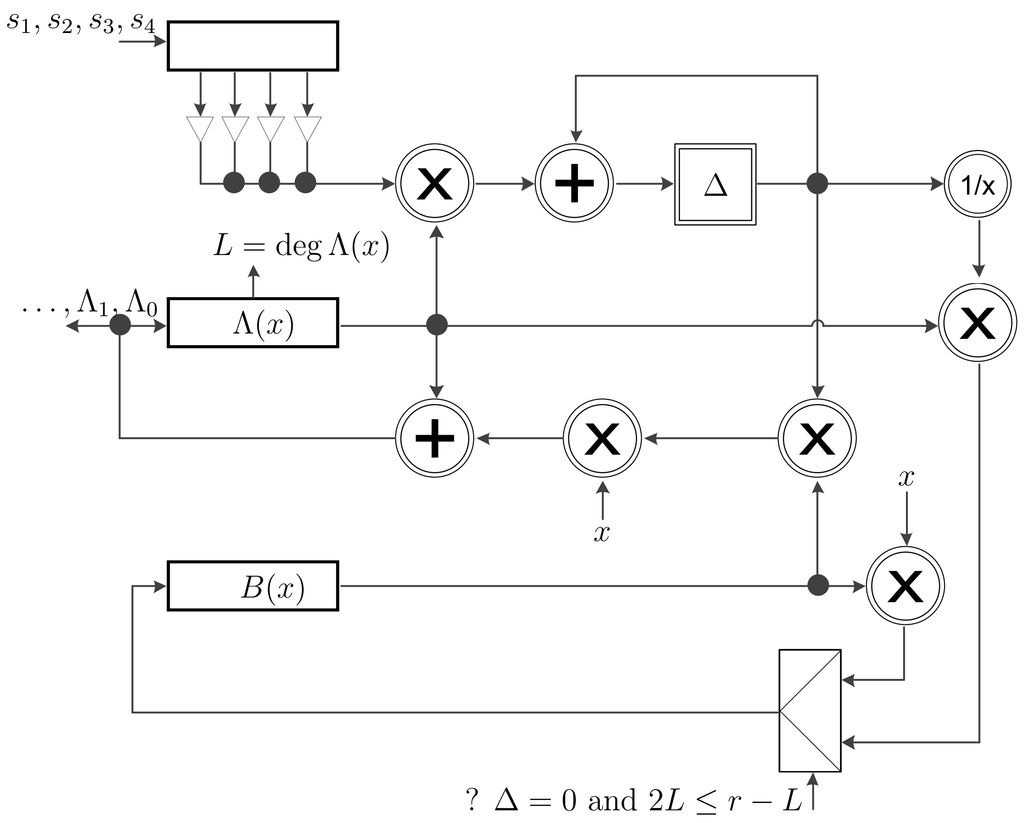


Figure 5.23: Berlekamp-Massey algorithm for $RS(n, n-4)$.

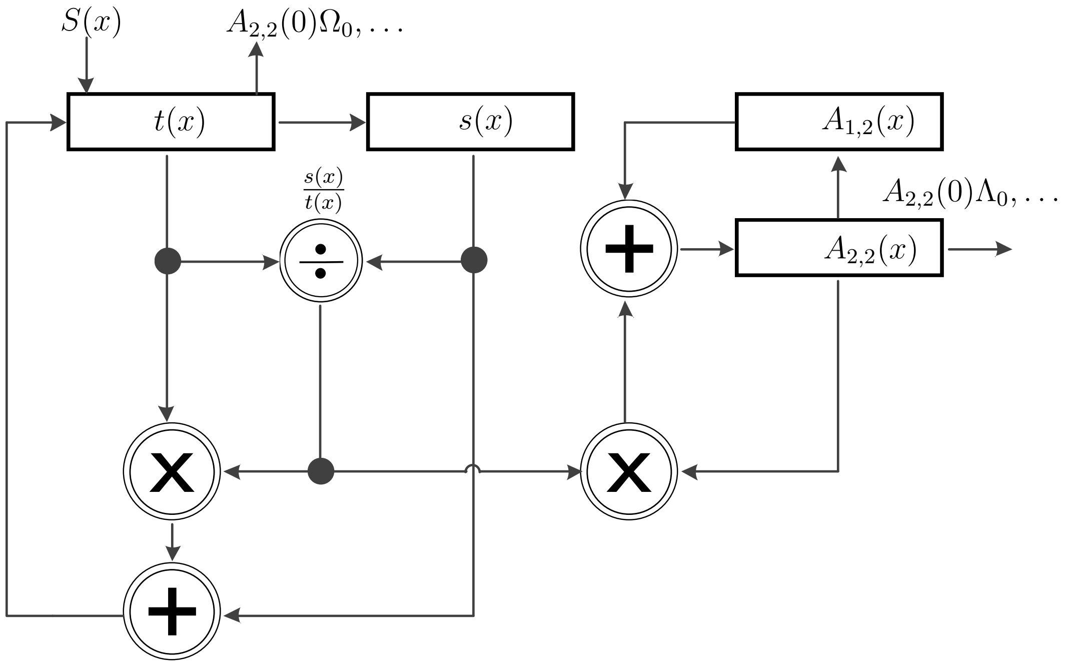


Figure 5.24: Sugiyama algorithm.

5.7 Chien search

Chien search is used to find error positions. For consecutive elements of Galois field $1, \ldots, \alpha^{q-2}$ there is checked whether evaluated element is root of error locator polynomial. On picture 5.25 is shown design of Chien search. It was mentioned ealier, that element Λ_0 is always 1, so it can be added to value of polynomial for $\Lambda_1, \ldots$.

5.8 Forney algorithm

Forney algorithm is used to find error values. To determine error value, there is needed derivative of error locator polynomial $\Lambda'(x)$, error evaluator polynomial $\Omega(x)$ and error locators X_j. For Galois fields of characteristic 2 derivative of $\Lambda(x)$ can be computed as filling with zeroes all coefficients on even positions of polynomials and then division of polynomial by polynomial x:

$$\Lambda'(x) = \frac{\sum_{j=0} \Lambda_j x^j}{x}; \Lambda_j = 0 : \text{for } j \text{ even}$$

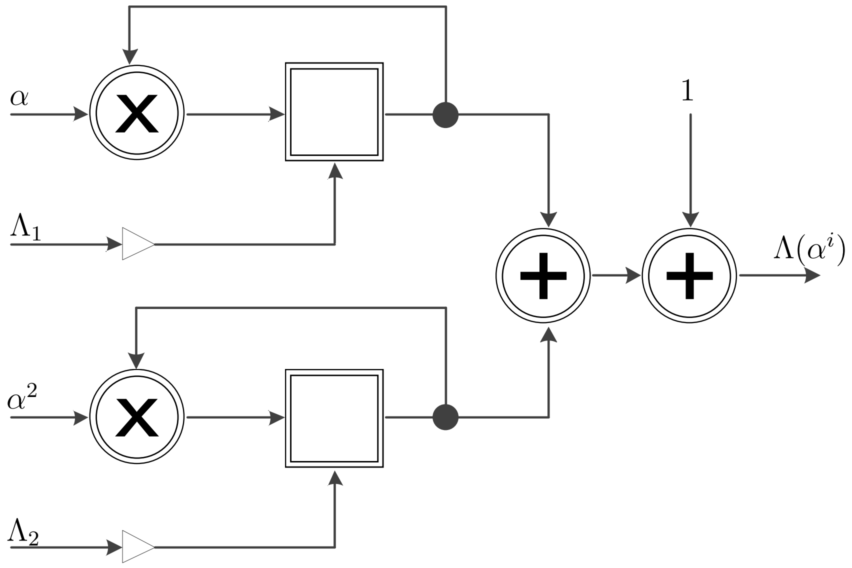


Figure 5.25: Chien search for $RS(7, 3)$.

Polynomial $\Omega(x)$ is as follows:

$$\Omega(x) \bmod x^{2t+1} = \Lambda(x)S(x)$$

It can be rewritten as:

$$\Omega(x) = \sum_{j=0}^{2t} D_j x^j, \qquad \text{where } D(x) = \Lambda(x)S(x)$$

Operation $\text{mod} x^{2t+1}$ is done by omitting all coefficients on positions greater than $2t$ of polynomial. Forney algorithm for $RS(7, 3)$ code is shown on picture 5.26. It computes error values for all elements in Galois field except zero. Forney algorithm equation is as follows:

$$e_{k_j} = -X_j \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})}$$

Values of polynomials $\Omega(x)$ and $\Lambda'(x)$ are computed for consecutive elements $1, \alpha, \ldots, \alpha^6$. Elements X_j are then as follows: $1, \alpha^6, \alpha^5, \ldots, \alpha$.

5.9 Guruswami-Sudan algorithm

Gurusuwami-Sudan algorithm decodes received vector by interpolation of polynomial of two variablies and then factorization of this polynomial.

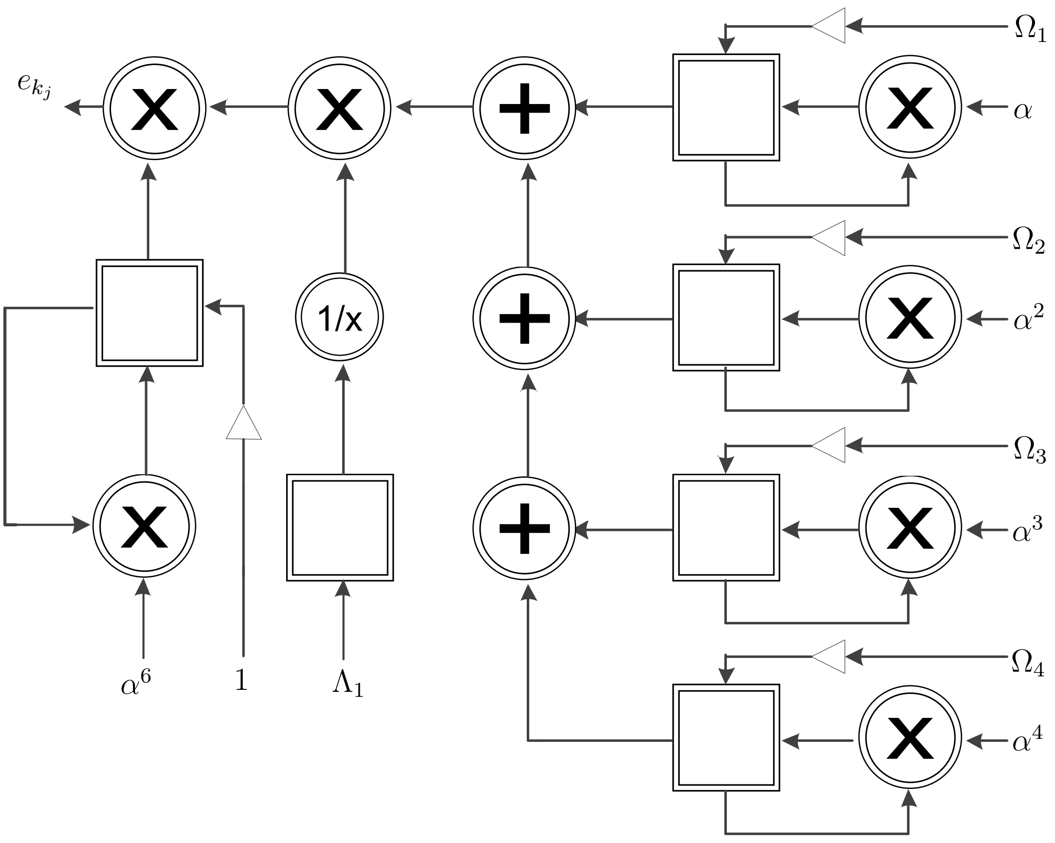


Figure 5.26: Forney algorithm for $RS(7, 3)$.

During implementation of Guruswami-Sudan algorithm there were written functions which are used to operate on polynomials of two variables, like determining degree of polynomial, evaluation of polynomial or multiplication of polynomial. Polynomials of two variables are ordered for variable y:

$$Q(x,y) = y^0(w_0(x)) + y^1(w_1(x)) + y^2(w_2(x)) + \dots$$

In order to find degree of polynomial for some order, there must be choosen monomial from this polynomial of greatest degree:

$$\deg_w Q(x,y) = \max\{\deg_w y^i(w_i(x))\}$$

To check if polynomial $Q(x,0)$ is equal to zero, then there must be check whether polynomial $w_0(x)$ is equal to zero.

To determine error correction capability of algorithm and maximal number of returned results, there must be known positions of monomial of type x^i and y^j on ordered list of monomials. Such list can be created by listing monomials x^iy^j in some order $(1,k-1)$ w-revlex. To find position of some monomial x^iy^j on this list, there is done simple search and when monomial is found then its position is returned. Size of list increases together with k. Furthermore, this list contains many monomials which are not needed, so place is wasted.

It can be observed positions of monomials x^i are regular and they generate a sequence. For order $(1,2)$ w-revlex:

$$Ind(x^0) = 0,$$

$$Ind(x^1) = 1(+1), Ind(x^2) = 2(+1),$$
$$Ind(x^3) = 4(+2), Ind(x^4) = 6(+2),$$
$$Ind(x^5) = 9(+3), Ind(x^6) = 12(+3),$$
$$Ind(x^7) = 16(+4), Ind(x^8) = 20(+4),$$
$$Ind(x^9) = 25(+5), Ind(x^{10}) = 30(+5),$$
$$Ind(x^{11}) = 36(+6), Ind(x^{12}) = 42(+6)$$

For order $(1, 4)$ w-revlex:

$$Ind(x^0) = 0,$$

$$Ind(x^1) = 1(+1), Ind(x^2) = 2(+1), Ind(x^3) = 3(+1), Ind(x^4) = 4(+1)$$

$$Ind(x^5) = 6(+2), Ind(x^6) = 8(+2), Ind(x^7) = 10(+2), Ind(x^8) = 12(+2)$$

$$Ind(x^9) = 15(+3), Ind(x^{10}) = 18(+3), Ind(x^{11}) = 21(+3), Ind(x^{12}) = 24(+3)$$

Function which determines position of monomial x^j on ordered list for order $(1, k-1)$ w-revlex is as follows:

```
Dane wejściowe: order (1, k - 1) w-revlex,
                exponent j of monomial x^j
int order = 0;       //position on the list
int counter = k - 1; //variable which is used to increase
                     //value which is added to position
                     //during iterations
int add_value = 1;   //value which is added to position
                     //during iterations
for(int i = 0; i < j; i++){
   order = order + add_value;
   counter--;
   if(counter == 0){
      counter = k - 1;
      add_value = add_value + 1;
   }
}
return order;
```

For monomials of type y^j first positions on the list of order $(1, 4)$ w-revlex are as follows:

$$Ind(y^0) = 0, Ind(y^1) = 5(+1+4), Ind(y^2) = 14(+1+4+4),$$

$$Ind(y^3) = 27(+1+4+4+4), Ind(y^4) = 44(+1+4+4+4+4),$$

$$Ind(y^5) = 65(+1+4+4+4+4+4)$$

For order $(1, 2)$ w-revlex:

$$Ind(y^0) = 0, Ind(y^1) = 3(+1+2), Ind(y^2) = 8(+1+2+2),$$

$$Ind(y^3) = 15(+1+2+2+2), Ind(y^4) = 24(+1+2+2+2+2),$$

$$Ind(y^5) = 35(+1+2+2+2+2+2)$$

```
Dane wejściowe: order (1, k - 1) w-revlex,
                exponent j of monomial y^j
if(j == 0)
   return 0;
int add_value = k;  //value which is added to position
                    //during iterations
int order = 0;      //position on the list
for(int i = 0; i < j; i++){
   order = order + add_value;
   add_value = add_value + k - 1;
}
return order;
```

6. Evaluation

6.1 Simulation setup

Implemented algorithms are tested in order to evaluate time of decoding, because this variable describes mostly computional complexity of algorithms:

- number of errors in received vectors,
- length of codeword,
- number of control elements,
- value of order of multiplicity of zeroes (Guruswami-Sudan algorithm),
- noise intensity in received vector (soft Guruswami-Sudan algorithm - Kötter-Vardy algorithm).

The focus was mainly on testing of software implementation. To check how much time decoders were working, there was used class CStopWatch [20]. To simulate algorithms implemented in hardware, there was used iSim simulator, which is part of Xilinx ISE Webpack. Results of tests and conclusions are written in the next part of this chapter.

There were done 100 iterations for each experiment case and mean from returned values is given as final result. Simulator lets to specify following properties of code:

- length of codeword,
- field used to create code,
- number of control elements,
- number of occured errors,
- noise in communication channel.

All independent variables except noise in communication channel are applied to all decoding algorithms. Noise in communication is applied only for soft decoding Guruswami-Sudan algorithm, because only this algorithm can use soft information.

Returned results from decoders are compared with original codeword in order to check whether decoding process was correct. There is also returned status information which determines if received vector was without errors or there occured errors and they were corrected or there occured errors and they cannot be corrected. Another variable which describes decoding process is time of decoding.

7. Results and discussion

7.1 Cyclic decoding

Cyclic decoding is done by searching some remainder of division of received vector by code generator polynomial. To find appropriate remainder, sometimes received vector must be shifted some times and division must be processed. If there is the same probability that errors occur on some position, then average there must be done about $\frac{n}{2}$ shifts for $RS(n,k)$, so decoding takes more time with increase of length of codeword.

Degree of code generator polynomial is associated with error correction capability. If error correction capability is higher, then there must be done division on longer polynomials. Remainder from division covers more elements in received vector when error correction capability increases. For small codes like $RS(7,5)$ or $RS(7,3)$ time of decoding can differ much, but with increase of information elements in code, the difference is less. On picture 7.4 is shown graph for 3 codes of constant codeword length, but variable information elements in codeword. With increase of information elements, degree of code generator polynomial is smaller, and thus, division takes more time.

Cyclic decoder in hardware implementation consists of 4 parts:

- syndrome register,
- buffer register,
- unit which checks Hamming weight of syndrome,
- control unit.

Syndrome register becomes bigger together with error correction capability. It consists of division unit with $2t$ registers for Galois field elements, at least $2t$ multiplication elements by constant and $2t$ addition elements. With increase divisor by 1, there is added 1 register for Galois field element, 1 multiplication element and 1 addition element.

Buffer register consists of k registers for Galois field elements. It becomes bigger linearly with increase of information elements in code.

Unit which checks Hamming weight of syndrome must recognize more syndrome patterns when error correction capability increases, so unit becomes more complicated.

Cyclic decoding has a drawback, that errors must be contained within $2t$ consecutive positions on received vector. For 1 error patterns it is no constraint, but for 2 error patterns this is possible that no error patterns will be corrected, altough code theoreticaly should correct there errors. On graphs 7.1, 7.2 and 7.3 there are shown times of decoding for 3 codes $RS(n,k)$ and amount of received vectors which were decoded not correctly. It can

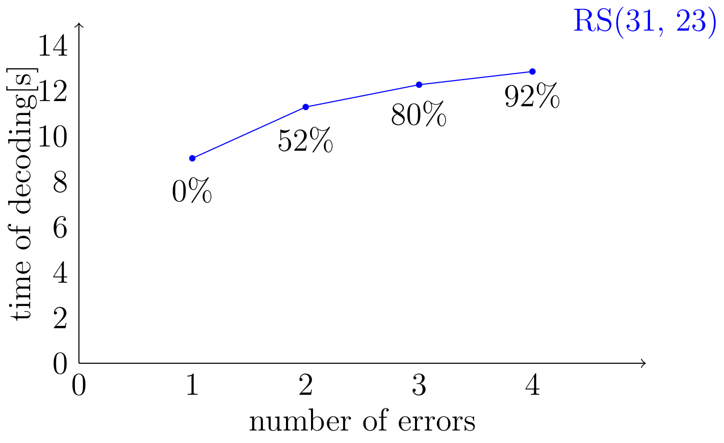


Figure 7.1: Cyclic decoding for RS(31, 23) with amount in % of not corrected received vectors.

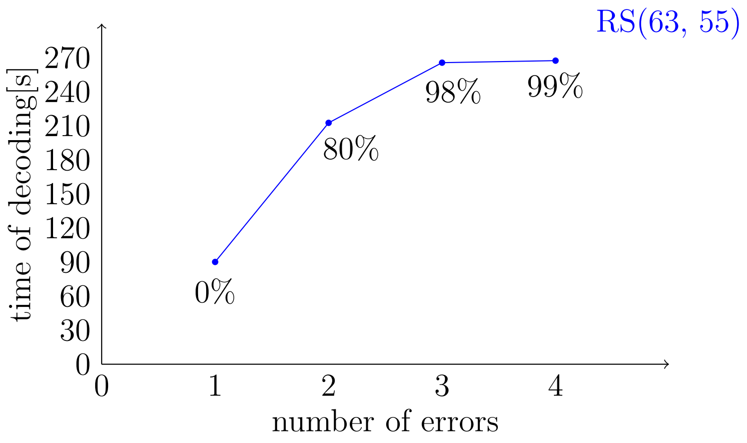


Figure 7.2: Cyclic decoding for RS(63, 55) with amount in % of not corrected received vectors.

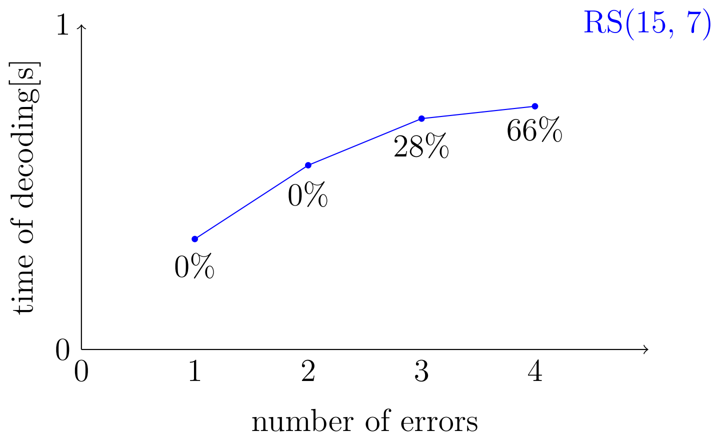


Figure 7.3: Cyclic decoding for RS(15, 7) with amount in % of not corrected received vectors.

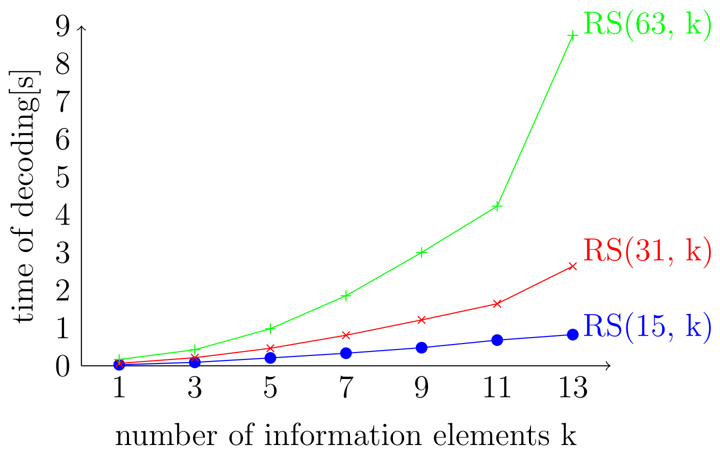


Figure 7.4: Cyclic decoding for RS(15, k), RS(31, k), RS(63, k).

be observed, that decoding time for $RS(63, 55)$ is about 10 times longer than for code $RS(31, 23)$ and about 90 times longer than for $RS(15, 7)$ code. All of these codes have the same error correction capability. The factor which influences time of decoding is that for longer codes there must be cone more shifts to find appropriate remainder from division, and for longer codes division is done on longer polynomials. It can be also observed that amount of not correctly decoded received vectors increases with length of codeword. For example for $RS(15, 7)$ there doesn't exist error pattern with 1 or 2 errors which cannot be correct, but it doesn't correct all 3 error patterns. It is worse for longer codes, $RS(65, 55)$ cannot correct about 80% of error patterns with 2 errors, however theoretically it can correct up to 4 errors.

Decoding can be boosted with use of multithreading. In the worst case there must be done $n - 1$ shifts in order to correct received vector. Divisions of polynomials can be split between 2 or more threads. On graph 7.5 there is shown that two threads can speed up decoding.

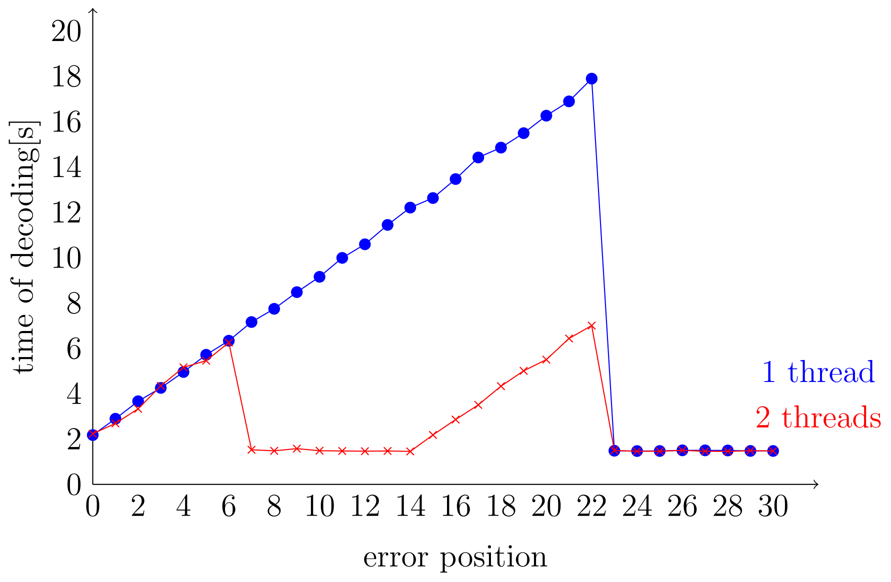


Figure 7.5: Cyclic decoding with multithreading.

7.2 Computing error positions

To compute error positions are used following 3 algorithms:

- Peterson-Gorenstein-Zierler algorithm,
- Berlekamp-Massey algorithm,

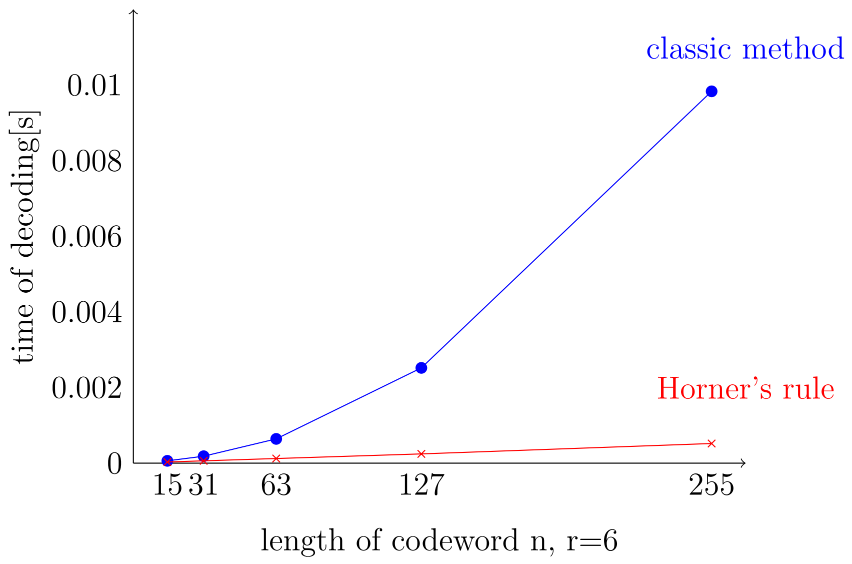


Figure 7.6: Computing syndromes with classic method and with Horner's rule.

- Sugiyama algorithm with erasures and without erasures.

These algorithms can correct number of errors up to theoretical error correction capability.

Input data for algorithms are $2t$ syndromes. Syndromes can be computed with classic method by multiplying all coefficients of received vector by powers of variable and sum of all products. Horner's rule can be used to reduce number of multiplications and boost syndrome computing. On graph 7.6 is shown difference between computing syndromes with classic method and with Horner's rule. Syndromes can be computed with use of sequential and combinational units.

7.2.1 Peterson-Gorenstein-Zierler algorithm

Peterson-Gorenstein-Zierler algorithm consists of two main steps: computing number of errors and finding positions of errors. Computing number of errors can be done by computing determinant of matrix. There was used Laplace expansion for this task with complexity $O(n!)$, but there exist faster algorithms to solve this task for example Sarrus method. Next, there is solved set of equations in order to find positions of errors. There was used Gaussian elimination for this task. On picture 7.7 is shown graph for code $RS(31, 31 - 2t)$, where error correction capability is variable. Time of de-

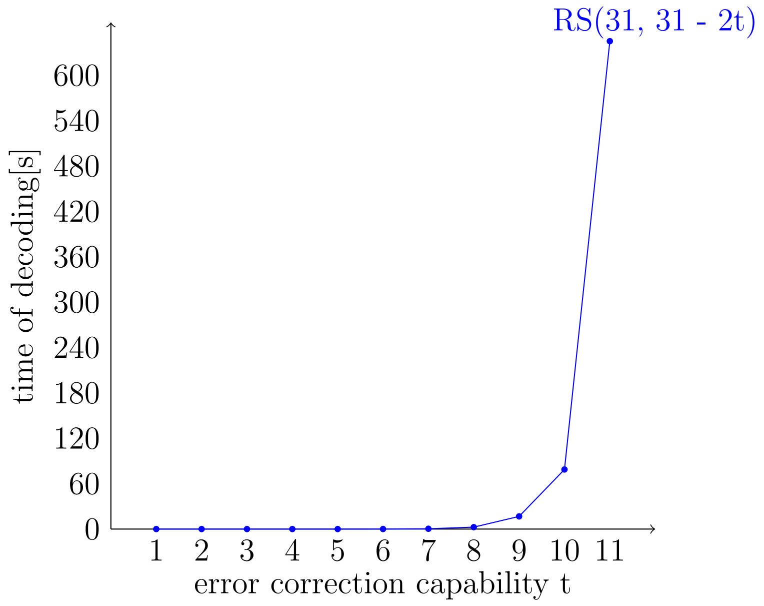


Figure 7.7: PGZ algorithm and error correction capability.

coding up to $t = 8$ is constant in presented time dimension, but for $t > 8$ it increases fast.

For codes of small error correction capability there can be designed not much complicated decoders. Example of design of such decoder was presented in last chapter. Error correction capability influences complexity of decoder. To compute roots of error locator polynomial, there can be used combinational units for polynomial evaluation. For $RS(n,k)$ there must be checked $n-1$ elements of $GF(n+1)$, so there must be used $n-1$ units of combinational polynomial evaluation, what becomes problem for longer codes. There must be also used n addition elements which add error vector with received vector in order to correct errors.

7.2.2 BMA algorithm and Sugiyama algorithm

Berlekamp-Massey algorithm within $2t$ iterations constructs error locator polynomial. Number of iterations is constant and is not dependent from number of occured errors. Sugiyama algorithm computes error locator polynomial within e iterations, where e denotes number of errors, so number of iterations is variable. With increase of error correction capability, Berlekamp-Massey algorithms needs more iterations to construct error locator polynomial, and Sugiyama algorithms employs longer polynomials in division. On picture 7.8 is shown how error correction capability affects $(63, 63-2t)$ code decoding

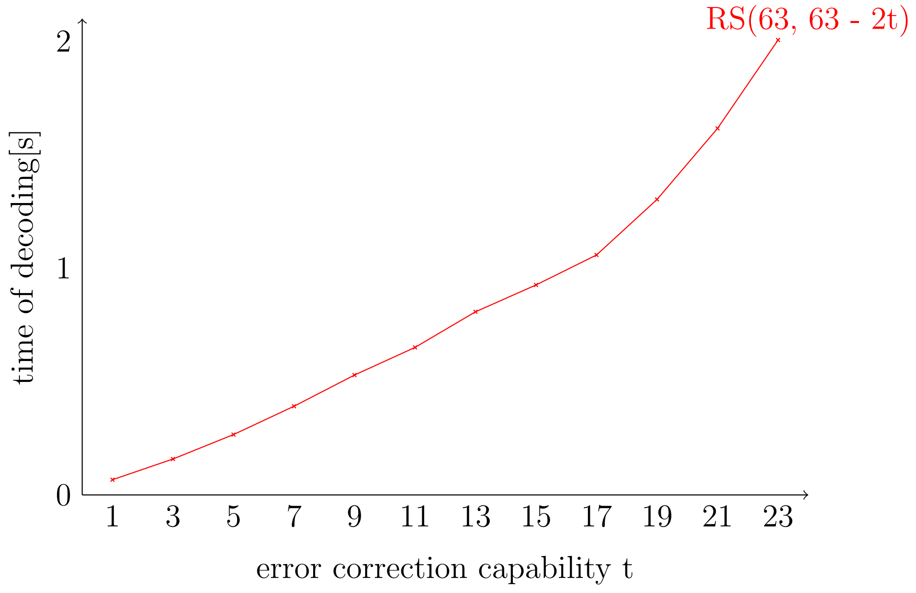


Figure 7.8: BMA algorithm for codes of variable error correction capability.

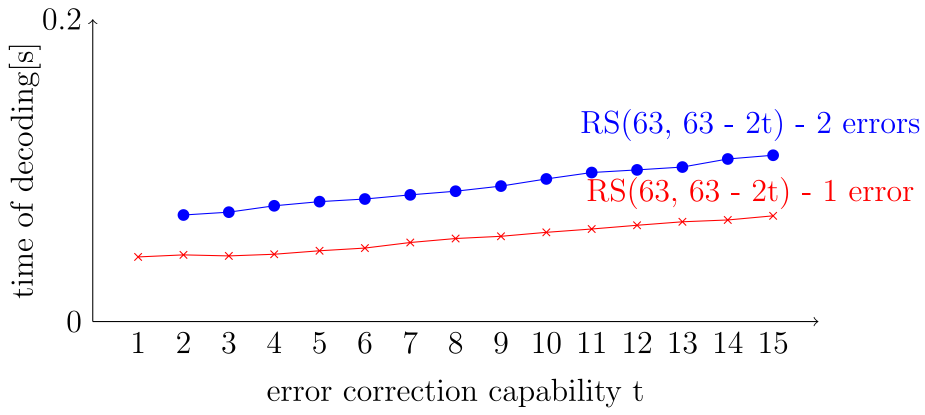


Figure 7.9: Sugiyama algorithm for RS codes of variable error correction capability.

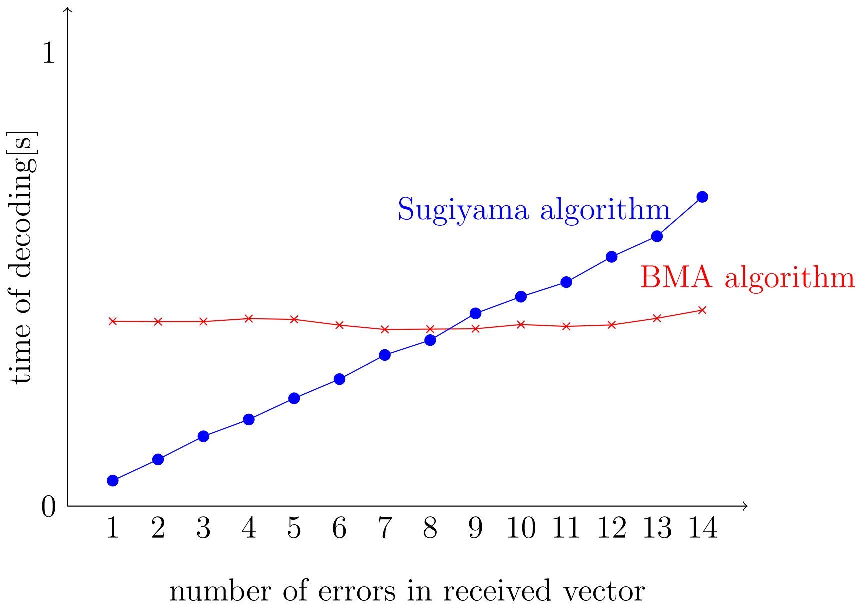


Figure 7.10: Berlekamp-Massey algorithm and Sugiyama algorithm for RS(63,3) and variable number of errors in received vector.

for Berlekamp-Massey algorithm, and on picture 7.9 is shown Sugiyama algorithm for code of variable error correction capability and for the cases of 1 and 2 errors. It can be observed, that increase of error correction capability influences much Berlekamp-Massey algorithm, but not Sugiyama algorithm. However decoding time for Sugiyama algorithm is dependent from number of occured errors. On picture 7.10 are compared Berlekamp-Massey and Sugiyama algorithms for variable number of errors. It can be observed to for small number of errors Sugiyama algorithm works faster, but time of decoding increases together with number of errors. For Berlekamp-Massey algorithm time is constant.

It can be observed that for next iterations of Berlekamp-Massey algorithm there are needed consecutive syndromes $s_1, s_2, \ldots, s_{2t}$. In first iteration there is needed s_1, in second s_1 and s_2 and so on. Decoder can work in parallel with syndrome computation unit. If iteration in Berlekamp-Massey algorithm lasts for 1 cycle, then syndrome computation and error locator computation will take $2t + 1$ cycles.

Sugiyama algorithm returns error locator polynomial and also error evaluator polynomial. Berlekamp-Massey returns only error locator polynomial, so there must be still computed error evaluator polynomial.

Time decoding can be speeded up by use of erasures. It requires that re-

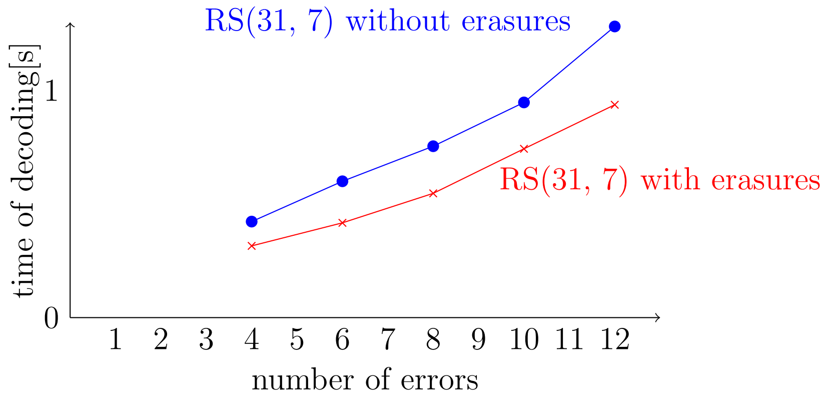


Figure 7.11: Comparision of decoding with erasures and without erasures. For case with erasures, number of true errors (one symbol was changed to another symbol) is equal to number of errors, and for case without erasures number of errors and erasures is equal to half of number of errors.

ceivers understund erasures. Erasures give additional knowledge, it is known where errors occured before starting of decoding. Moreover, more erroneous positions can be corrected. There was implemented modified Sugiyama algorithm for erasures. Decoding is faster as it is shown on picture 7.11. For the same number of erroneous positions, there are needed less iterations. However algorithm is a bit more complicated.

7.3 Computing error values

Error values can be computed by solving set of equation with use of Gaussian elimination. Another method is to use Forney algorithm. Forney algorithm uses derivative of error locator polynomial and error evaluator polynomial. It is computed for each error position. On picture 7.12 there are compared Forney algorithm and Gaussian elimination. Forney algorithm solves problem by evaluation of polynomial, Gaussian eliminations solve sets of equations. Gaussian elimination can be better for codes of small error correction capability, but Forney algorithm has advantage when codes are longer.

Forney algorithm can work in parallel with Chien search. In the same time are computed error positions and error values. Chien search determines whether position is erroneous or not.

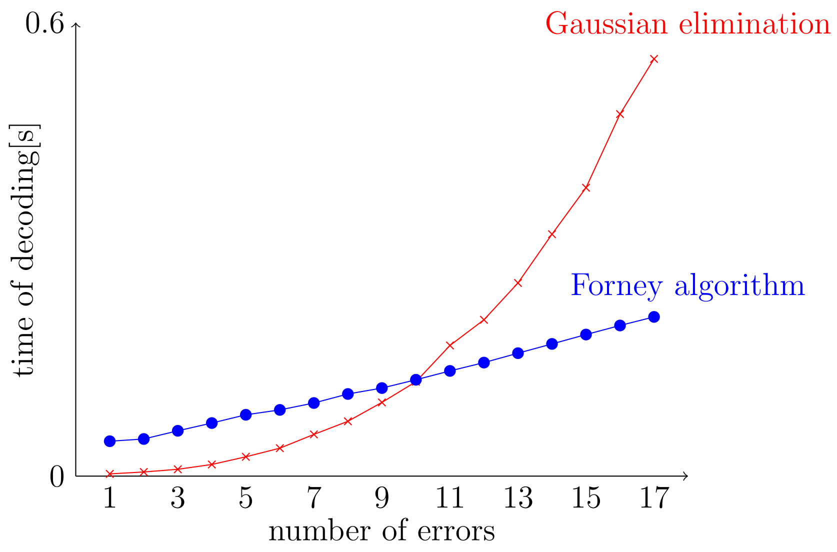


Figure 7.12: Comparision of Gaussian elimination and Forney algorithm.

7.4 Guruswami-Sudan algorithm

Guruswami-Sudan algorithm consists of two steps:

- interpolation step,
- factorization step.

Interpolation step is solved by Kötter algorithm. Its complexity is dependent from number of points which are used for interpolation and from order of multiplicity of zeroes for these points.

Order of multiplicity of zeroes influences error correction capability of algorithm and also time of decoding. On graph 7.13 is shown how order of multplicity of zeroes affects error correction capability. It can be observed, that for smaller values of m it is easier to improve error correction capability for codes with greater number of control elements. For codes with less number of control elements improvement is worse.

On picture 7.14 is shown how order of multiplicity of zeroes affects time of decoding. When parameter m increases, then time of decoding increases also. For constant m for all points complexity for Guruswami-Sudan algorithm is $O(n^2m^4)$ [7].

Kötter-Vardy algorithm is used to transform soft information to hard information, which is then used by Guruswami-Sudan decoder.

Example of corrected vector for original codeword of $RS(7,3)$:

$$c(x) = \alpha^5x^6 + \alpha^6x^4 + \alpha x^2 + \alpha^6x + \alpha^5$$

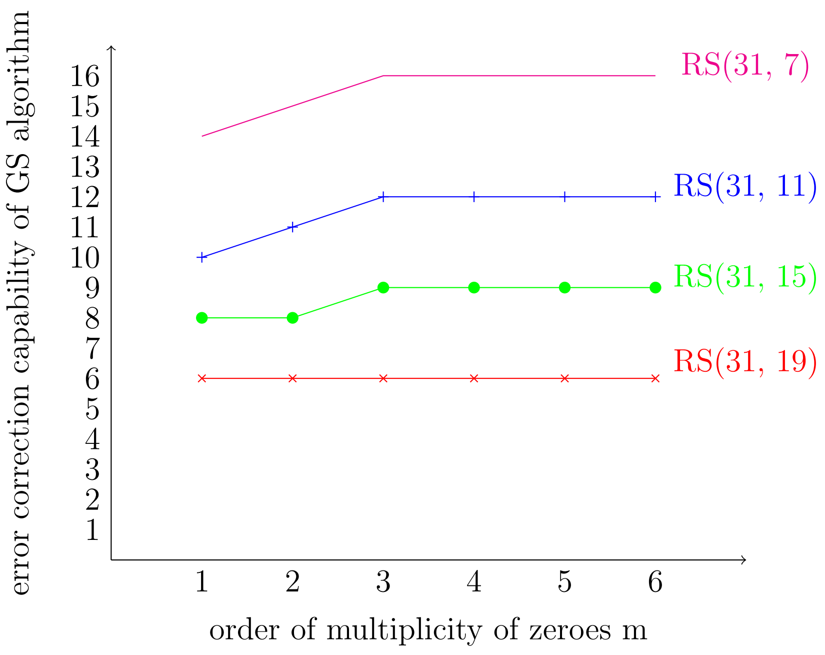


Figure 7.13: Error correction capability and multiplicity of zeroes.

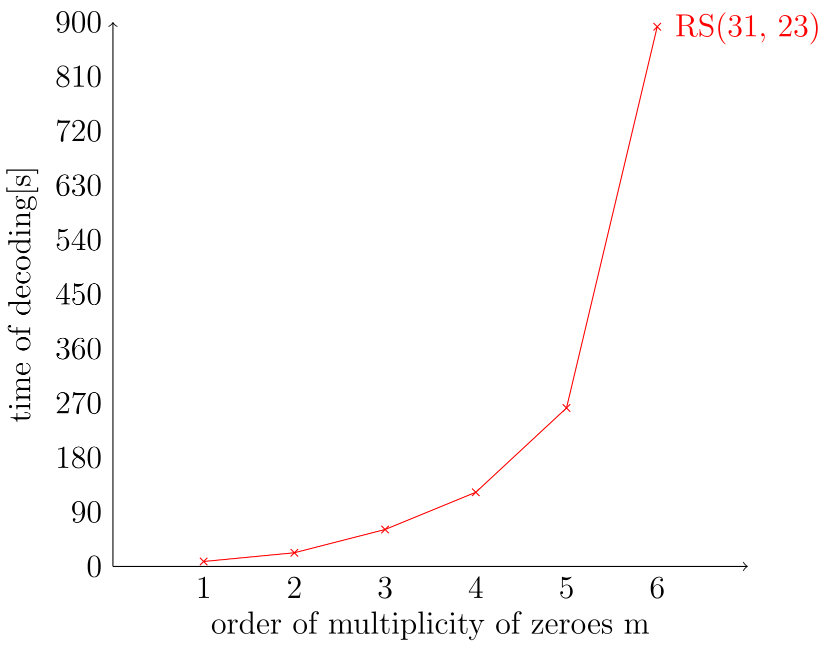


Figure 7.14: Time of decoding and order of multiplicity of zeroes.

Received vector with 6 errors:

$$r = ((-0.718, -0.958, -1.926), (-1.203, -1.894, -0.145),$$

$$(0.799, -0.033, -0.868), (-0.269, 0.965, 0.825), (0.054, -1.471, 1.874),$$

$$(-0.281, 2.204, 1.537), (-2.24, -1.753, 0.013))$$

Transformation of soft received vector to hard received vector, that values < 0 are changed to 1, and values ≥ 0 are changed to 0:

$$r(x) = \alpha^6 x^6 + x^5 + \alpha^2 x^4 + x^3 + \alpha^4 x^2 + \alpha^5 x + \alpha^5$$

Result of Guruswami-Sudan algorithm:

$$w_1(x) = \alpha^3 x^2 + \alpha^6 x + 1$$

$$w_2(x) = x^2 + \alpha^2 x + \alpha$$

Algorithm was able to crrect 6 errors in received vector of $RS(7, 3)$ code. Possibility of error correcting depends on noise which disrupted codeword. Algorithm works good, if given value is more or less equally between -1 and 1. Furthermore, modulation can affect decoding and also method which computes reliability matrix. Error correction capability of algorithm is variable and it depends on occured errors. During tests for $RS(7, 3)$ code algorithm could correct all 3 error patterns, often 4 errors patterns, and seldom error patterns with greater number of occured errors.

7.5 Summary

Summarizing this chapter, there are given answers for research questions stated before writing master thesis:

Question: What are the features of scalability and complexity of implemented algorithms?
Answer: Cyclic decoding algorithm is not always able to correct all valid error patterns, however theoretically it should be able to do it. Errors must be contained within $2t$ consecutive positions of received vector in order to be corrected. Cyclic decoding algorithm is able to correct all error patterns of 1 error. Time of decoding depends on error correction capability of code and length of codeword. With increase of error correction capability there must be executed less divisions in order to decode received word. With increase of codeword length there must be executed on average more divisions and division takes more time, because dividend is of higher degree.

Peterson-Gorenstein-Zierler algorithm is able to correct all error patterns independently from error positions. It is not practical (due to time of decoding in software implementation) for codes of high error correction capability (for tests it was $t > 8$), because time of decoding increases fast as error correction capability increases. For codes of small error correction capability (for tests $t = 2$), there can be constructed efficient decoders in hardware. For codes of higher error correction capability decoders become much more complex, because additional circuits are needed. Time of decoding depends on error correction capability and number of errors (software implementation), but it doesn't depend on length of codeword and number of information elements. With increase of error correction capability there must be computed determinants of bigger matrices. If number of errors decreases, then there must be computed determinents of higher number of matrices. Complexity of algorithm is dependent from method of computing determinant of matrix and from method used to solve set of equations.

Berlekamp-Massey algorithm decodes received vector within constant number of iterations, which is dependent from error correction capability. Time of decoding depends on error correction capability. It doesn't depend on length of codeword, number of occurred errors and number of information elements. Algorithm decodes received vector within $2t$ iterations.

Sugiyama algorithm decodes received vector within variable number of iterations, which is determined by number of occurred errors in received vector. Algorithm computes also error evaluator polynomial. Sugiyama algorithm employs division of polynomials, which is time-consuming operation. Time of decoding depends on number of occurred errors in received vector and on error correction capability, it doesn't depend on length of codeword, number of information elements. With increase of number of errors there must be executed more iterations of division of polynomials.

Guruswami-Sudan algorithm is able to correct more errors than it theoretically should be. Time of decoding depends on length of codeword and order of multiplicity of zeroes.

Question: What mathematical functions and modifications on algorithm level can improve performance of decoding algorithms?

Answer: Addition in Galois fields can be done in real time with use of Zech logarithms. Imamura algorithm can be used to compute values of Zech logarithms.

Using erasures can cause that decoder is able to correct more erroneous positions in received vector. However decoder is then more complicated, but on average it can decode faster.

Computing of orders of monomials of type x^i and y^j can be described as problem of computing next values of arithmetic progression.

Question: Which features of software and hardware can improve performance of decoding algorithms?
Answer: Multithreading can speed up cyclic decoding. Decoding process can be done by independently working threads. Decoding process is finished when one of these threads manages to decode received vector.

Addition and multiplication can be implememented with using LUT tables or computing results in real time. LUT tables dont't require implementation of arithmetic functions for Galois fields, but they require memory. Computing results in real time requires implementation of Galois field arithmetic, but it requires less memory, just to store Zech logarithms.

Berlekamp-Massey algorithm can work in parallel with syndrome computing unit. If one iteration in Berlekamp-Massey algorithm takes time of one cycle and syndrome unit computes syndrome value within one cycle, then computation of error locator polynomial takes $2t + 1$ cycles.

8. Conclusions and future work

There are several decoding algorithms of Reed-Solomon code which have different approaches for the same task. It's hard to say which algorithm is the best at all, for example Guruswami-Sudan algorithm can correct more errors than other algorithms, but is also of biggest complexity, Peterson-Gorenstein-Zierler algorithm can be efficiently implemented for small codes, Berlekamp-Massey algorithm works constant time for any valid number of errors, but work of similar algorithm, which is Sugiyama algorithm, is dependent from number of errors. Each of evaluated algorithms have some advantages and disadvantages. Evaluation of algorithms gives answer for choice of decoding algorithm according to system requirements.

Except evaluated decoding algorithms there some more which weren't mentioned here like Welch-Berlekamp algorithm [23], GMD algorithm [22] or Jiang-Narayanan algorithm [21]. Not all algorithms were implemented in hardware, furthermore evaluated were codes with length up to 255. These things can be objects of future researches.

Bibliography

[1] I. S. Reed, G. Solomon, *Polynomial Codes Over Certain Finite Fields*, Journal of Society for Industrial and Applied Mathematics, vol.8, no.2, pp.300-304, 1960.

[2] W. W. Peterson, *Encoding and Error-Correction Procedures for the Bose-Chaudhuri-Codes*, Information and Theory, vol.6, no.4, pp.459-470, 1960.

[3] D. Gorenstein, N. Zierler, *A Class Of Error-Correcting Codes In* p^m *Symbols*, Journal of the Society for Industrial and Applied Mathematics, vol.9, no.2, pp.207-214, 1961.

[4] E. Berlekamp, *Non-binary BCH Decoding*, Information Theory, IEEE Transactions on, vol.14, no.2, p.242, 1968.

[5] J. L. Massey, *Shift-Register Synthesis and BCH Decoding*, Information Theory, IEEE Transactions on, vol.15, no.1, p.122, 1969.

[6] R. Singleton, *Maximum distance q-nary codes*, Information Theory, IEEE Transactions on, vol.10, no.2, pp.116-118, 1964.

[7] R. J. McEliece, *The Guruswami-Sudan decoding algorithm for Reed-Solomon codes*, IPN Progress Report, Tech. Rep. 42–153, 2003.

[8] S.-W. Lee, B. V. K. V. Kumar, *Application of Soft-Decision Decoders to Non Narrow-Sense Reed-Solomon Codes*, Communications, IEEE International Conference on, 2007.

[9] W. Mochnacki, *Kody korekcyjne i kryptografia*, Oficyna Wydawnicza Politechniki Wrocławskiej, 2000.

[10] J. Biernat, Wykłady z przedmiotu *Kodowanie i szyfrowanie danych*, Politechnika Wrocławska, 2009.

[11] T. Jurlewicz, Z. Skoczylas, *Algebra liniowa 1. Definicje, twierdzenia, wzory*, Oficyna Wydawnicza GiS, 2000.

[12] T. K. Moon, *Error correction coding: mathematical methods and algorithms*, 2005.

[13] R. T. Chien, *Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes*, Information Theory, IEEE Transactions on, vol. 10, no.4, pp.357-363, 1964.

[14] R. E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press, 2003.

[15] R. E. Blahut, *Algebraic Codes on Lines, Planes and Curves*, Cambridge University Press, 2008.

[16] K. Imamura, *A method for computing addition tables in $GF(p^n)$*, Information Theory, IEEE Transcations on, vol.26, no.3, p.367, 1980.

[17] L. Chen, R.A. Carrasco, E.G. Chester, *Decoding Reed-Solomon codes using the Guruswami-Sudan algorithm*, Proc. 5th IEEE International Symposium of Communication Systems, Networks and Digital Signal Processing (CSNDSP), 2006.

[18] R. A. Carrasco, M. Johnston, *Non-binary error control coding for wireless communication and data storage*, Wiley, 2008.

[19] R. Koetter, A. Vardy, *Algebraic Soft-Decision Decoding of Reed-Solomon Codes*, Information Theory, IEEE Transactions on, vol.49, no.11, pp.2809-2825, 2003.

[20] D. Bolton, *How do I do High Resolution Timing in C++ on Windows?*, *http://cplus.about.com/od/howtodothingsi2/a/timing.htm*

[21] J. Jiang, K. R. Narayanan, *Iterative Soft Decoding of Reed-Solomon Codes*, IEEE Communications Letters, vol.8, no.4, pp.244-246, 2004.

[22] G. Forney, *Generalized minimum distance decoding*, Information Theory, IEEE Transactions on, vol.12, no.2, pp. 125-131, 1966.

[23] L. R. Welch, E. Berlekamp, *Error correction for algebraic block codes*, US Patent, Number 4,633,470, 1986.

[24] S.-F. Wang, H.-Y. Hsu, A.-Y. Wu, *A very low-cost multi-mode Reed Solomon decoder based on Peterson-Gorenstein-Zierler algorithm*, Signal Processing Systems, IEEE Workshop on, pp.37-48, 2001.

[25] S. Czynszak, *Proposal for Master Thesis in Computer Science*, 2011.

[26] B. Vucetic, V. Ponampalam, J. Vuckovic, *Low Complexity Soft Decision Decoding Algorithms for Reed-Solomon Codes*, IEICE Trans. Commun., Vol. E84-B, No. 3, 2001.

[27] H.-C. Chang, C. B. Shung, New Serial Archtecture for Berlekamp-Massey Algorithm, IEEE Transactions on Communications, Vol. 47, No. 4, 1999.