VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SROVNÁNÍ ALGORITMŮ DEKÓDOVÁNÍ REED-SOLOMONOVA KÓDU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JIŘÍ ŠICNER
AUTHOR

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SROVNÁNÍ ALGORITMŮ DEKÓDOVÁNÍ REED-SOLOMONOVA KÓDU

COMPARISON OF REED-SOLOMON DECODING ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ ŠICNER
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL ŠILHAVÝ, Ph.D.
SUPERVISOR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Jiří Šicner	***ID:***	98535
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Srovnání algoritmů dekódování Reed-Solomonova kódu

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku protichybového zabezpečení pomocí Reed-Solomonova kódu. Dále se zaměřte na problematiku dekódování zprávy zabezpečené pomocí Reed-Solomonova kódu. Porovnejte jednotlivé přístupy. V programovém prostředí Matlab vytvořte demonstrační program, který bude možno využít jako výukovou pomůcku a rovněž jako analyzačního nástroje pro porovnání dosažených parametrů kódů. Program koncipujte tak, aby jej bylo možno využít jako výukovou pomůcku. Vytvořený program by měl názorně demonstrovat jednotlivé algoritmy dekódování zprávy. Program bude zahrnovat grafické rozhranní a jednotlivé položky v něm doplňte nápovědou. S pomocí vytvořeného programu realizujte zevrubné srovnání dosažených parametrů jednotlivých přístupů pro různé parametry kódů.

DOPORUČENÁ LITERATURA:

[1] Moon, T. K. Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, 2005, ISBN-13: 978-0070010697.
[2] Lin, S., Costello, D. J.. Error Control Coding: Fundamentals and Applications, second edition, Prentice Hall: Englewood Cliffs, NJ, 2005 ISBN: 0-13-042672-5.
[3] Skalar, B.. Digital Communications, Fundamentals and applications, Prentice-Hall, 2003, ISBN 0-13-084788-7.

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Pavel Šilhavý, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Práce se zabývá kódováním a dekódováním Reed-Solomonových kódů. Je zde obecně popsáno algebraické dekódování Reed-Solomonových kódů a následně podrobně popsány čtyři metody dekódování, konkrétně Berlekamp-Masseyův algoritmus, Euklidův algoritus, Peterson-Gorenstein-Zierleův algoritmus a přímá metoda. Tyto metody jsou zde pak porovnány a některé z nich jsou realizovány v programu Matlab.

KLÍČOVÁ SLOVA

Reed-Solomonův kód, Berlekamp-Masseyův algoritmus, Euklidův algoritmus, Peterson-Gorenstein-Zierleův algoritmus, Přímá metoda

ABSTRACT

The work deals with the encoding and decoding of Reed-Solomon codes. There is generally described algebraic decoding of Reed-Solomon codes, and then described four methods of decoding, namely Masseyův-Berlekamp algorithm, Euclidean algoritus, Peterson-Gorenstein-Zierleův algorithm and the direct method. These methods are then compared, and some of them are implemented in Matlab.

KEYWORDS

Reed-Solomon code, Berlekamp-Massey algorithm, Euclidean algorithm, Peterson-Gorenstein-Zierler algorithm, Direct Solution

ŠICNER, Jiří *Srovnání algoritmů dekódování Reed-Solomonova kódu*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 59 s. Vedoucí práce byl Ing. Pavel Šilhavý, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Srovnání algoritmů dekódování Reed-Solomonova kódu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Pavlu Šilhavému Ph. D., za užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne .....

.....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Tato diplomová práce se zabývá problematikou Reed Solomonových kódů a hlavně způsoby jejich dekódování. Cílem práce, jak ho definuje zadání je srovnat jednotlivé přístupy k dekódování a tyto realizovat v programu Matlab. Práce je rozdělena do dvou základních částí.

První část je teoretická, seznamuje čtenáře se základními vlastnostmi protichybových kódů, podrobněji se pak zabývá rozborem Reed Solomonova kódování. Následuje rozbor dekódování a uvedení některých základních přístupů k dekódování. Tyto metody jsou zde pak porovnány z několika hledisek.

Druhá část je praktická. Je zaměřena na implementaci vybraných metod do programu Matlab. Jsou zde uvedeny simulace závislosti chybovosti FER na E_B/N_0 pro různé kódy a dekódovací přístupy. Dále je v této části dokumentace pro vytvořený výukový a simulační program s grafickým rozhraním.

1 REED - SOLOMONOVY KÓDY

Při přenášení dat v komunikačních systémech díky potřebě neustále zvyšovat přenosové rychlosti se stále častěji setkáváme se vznikem chyb, které mají tendenci se shlukovat. Jedním ze způsobů ochrany jsou Reed - Solomonovy kódy (dále jen RS kódy), ty jsou schopny přenos dostatečně zabezpečit. Byly publikovány v roce 1960 svými vynálezci Irvingem Reedem a Gusem Solomonem v tehdejším časopisu o vědě Journal of the Society for Industrial and Applied Mathematics, od svých vynálezců také přebrali svůj název. Tato kapitola se bude zabývat základním popisem a vlastnostmi RS kódů. V této kapitole je čerpáno především z [2] [4] [8] a [14].

1.1 Důležité pojmy z lineární algebry

Operace spojené s kódováním pomocí RS kódů se uskutečňují pomocí prvků z Galoisova tělesa $GF(p^r)$, kde p je základ číselné soustavy a r je stupeň rozšíření Galoisova tělesa. Galoisovo těleso vychází z konečného tělesa Z_p, vzniká rozšířením konečného tělesa.[2]

1.1.1 Konečné těleso

Konečné těleso bývá značeno Z_p. Je to algebraická struktura určená počtem svých prvků. Struktura je tvořena zbytky po dělení celých kladných čísel prvočíslem p. Toto prvočíslo pak určuje základ číselné soustavy. Jako příklad konečného tělesa bych uvedl v technické praxi často používané těleso Z_2, které je tvořeno množinou prvků $0, 1$. V této množině jsou pak chápány operace součtu jako operace exclusive or (XOR) a operace násobení jako logický součin (AND). Dále je nutno uvést, že operace prováděné nad konečným tělesem musí splňovat asociativní, komutativní a distributivní zákony. [4]

1.1.2 Galoisovo těleso

Galoisovo těleso vychází z konečného tělesa. Jedná se o rozšíření konečného tělesa stupněm r. V Galoisově tělese je celkový počet prvků roven p^r. Pro oblast kódování

se pak používá pouze $GF(2^r)$.

Podle [2] lze Galoisovo těleso vyjádřit pomocí zbytků po dělení mnohočlenů ze Z_2 určitým mnohočlenem (vytvářecím mnohočlenem), pro který platí:

1. je nerozložitelný v uvažovaném okruhu mnohočlenů
2. je stupně r
3. dělí beze zbytku $(x^n - 1)$ a žádný jiný dvojčlen nižšího stupně tohoto tvaru

1.2 Důležité pojmy z kódování

V této podkapitole bude uvedeno několik důležitých pojmů z problematiky korekčních kódů.[14]

Blokové kódy – zabezpečovací proces je uveden v rámci jednoho jediného bloku, který vznikl oddělením určitého úseku signálových prvků. Tento úsek se označuje jako *kódové slovo*.

Systematické kódy – u systematického kódu se ve výstupním slově dají oddělit od sebe zabezpečovací a informační bity. Kdežto u nesystematických to nelze, protože výstupní slovo se odvodí od poloh jedniček a nul ve vstupním slově.

Lineární kódy – libovolnou kódovou kombinaci lze odvodit jako lineární kombinaci z ostatních kódových kombinací. To je umožněno tím, že matematický základ těchto kódů využívá prostředků lineární algebry. Jsou definovány vytvářecí maticí **G**, nebo v některých případech, jako jsou cyklické kódy, vytvářecím mnohočlenem $G(x)$.

1.3 Základní popis RS kódů

RS kódy patří do kategorie BCH (Bose-Ray-Chaudhuri) kódů a jedná se tedy o korekční, cyklické, lineární kódy se symboly, které jsou tvořeny m-bitovými posloupnostmi, kde m je libovolné celé kladné číslo s hodnotou větší než 2. Kód bývá běžně zadán parametry (n, k). Parametr k určuje, kolik symbolů vstupuje do kodéru a parametr n udává počet symbolů vystupujících z kodéru. Pro n a k pak podle [8] platí:

$$0 < k < n < 2^m + 2 \tag{1.1}$$

Pro nejběžnější RS(n, k) kód pak platí[8]:

$$(n, k) = (2^m - 1,\ \ 2^m - 1 - 2t), \tag{1.2}$$

kde t je označení pro korekční schopnost kódu a $n - k = 2t$ je počet symbolů pro zabezpečení.

Pro nebinární kódy je vzdálenost mezi dvěma kódovými slovy definována (analogově s Hammingovou vzdáleností) jako počet symbolů, ve kterých se posloupnost liší. Pro RS kódy je minimální kódová vzdálenost dána vztahem podle [8]:

$$d_{min} = n - k + 1 \tag{1.3}$$

Kód je schopen opravit jakoukoliv kombinaci t nebo méně chyb, kde t lze vyjádřit jako [8]:

$$t = \frac{d_{min} - 1}{2} = \frac{n - k}{2} \tag{1.4}$$

Než přejdeme k samotnému procesu kódování, je potřeba ještě definovat vytvářecí polynom. Ten je podle [8] definován jako:

$$g(X) = (X - \alpha)(X - \alpha^2)\dots(X - \alpha^{2t}) = \prod_{j=1}^{2t}(X - \alpha^j) = \sum_{j=0}^{2t} g_j X^j, \tag{1.5}$$

kde $\alpha^1, \alpha^2, \dots, \alpha^{2t}$ jsou prvky Galoisova tělesa.

1.4 RS kódování

Protože RS kódy mohou být systematické i nesystematické, používají se pro kódování dva přístupy. Nesystematické cyklické kodéry generují kódové slovo vynásobením vstupního polynomu $i(X)$ vytvářecím polynomem $g(X)$. Výstupní kódové slovo je tedy dáno [8]:

$$c(X) = i(X).g(X) \tag{1.6}$$

Kde koeficienty polynomů jsou prvky Galoisova pole $GF(2^m)$.

Pro systematické RS kódy pak platí vztah [8]:

$$c(X) = X^{n-k} \cdot i(X) + p(X) \tag{1.7}$$

podle výše uvedené definice máme:

$$Rem\left\{\frac{c(X)}{g(X)}\right\} = 0 \tag{1.8}$$

$$Rem\left\{\frac{X^{(n-k)} \cdot i(X) + p(X)}{g(X)}\right\} = 0 \tag{1.9}$$

$$Rem\left\{\frac{X^{(n-k)} \cdot i(X))}{g(X)}\right\} + Rem\left\{\frac{p(X)}{g(X)}\right\} = 0 \tag{1.10}$$

Jestliže posloupnost paritního polynomu $p(X)$ je menší než $n-k$ a posloupnost $g(X)$ je $n-k$, pak můžeme podle [8] psát:

$$Rem\left\{\frac{p(X)}{g(X)}\right\} = p(X) \tag{1.11}$$

a dosazením této rovnice do rovnice 1.10 a upravením dostaneme:

$$-\,Rem\left\{\frac{X^{(n-k)} \cdot i(X))}{g(X)}\right\} = p(X) \tag{1.12}$$

Dosazením 1.12 do 1.7 pak dostaneme konečný vztah pro systematické kódování:

$$c(X) = X^{n-k} \cdot i(X) + Rem\left\{\frac{X^{(n-k)} \cdot i(X))}{g(X)}\right\}. \tag{1.13}$$

1.4.1 Příklad principu kódování RS kódem

V této podkapitole bych pro lepší pochopení uvedl příklad na zakódování vstupní posloupnosti systematickým RS kódem.

Pro názornost vezmeme kód RS (7, 3) určený Galoisovým polem GF (2^3) nad polynomem $p_i(X) = 1 + X + X^3$, které je tvořeno prvky viz tabulka 1.1. Zpráva k přenesení bude $i = (010\ \ 110\ \ 111)$.

Exponenciální tvar	Polynomiální tvar	Binární tvar
0	0	0 0 0
α^0	1	1 0 0
α	α	0 1 0
α^2	α^2	0 0 1
α^3	$1+\alpha$	1 1 0
α^4	$\alpha+\alpha^2$	0 1 1
α^5	$1+\alpha+\alpha^2$	1 1 1
α^6	$1+\alpha^2$	1 0 1

Tab. 1.1: Galoisovo pole generované polynomem $p_i(X) = 1 + X + X^3$

Nejprve je nutno určit vytvářecí mnohočlen $g(X)$. Protože máme kód RS (7, 3), víme, že bude podle vztahu 1.4 $t = 2$ a tedy můžeme vypočítat podle vztahu 1.5 vytvářecí mnohočlen pro náš kód.

$$\begin{aligned} g(X) &= (X-\alpha)(X-\alpha^2)\dots(X-\alpha^{2t}) \\ g(X) &= (X-\alpha)(X-\alpha^2)(X-\alpha^3)(X-\alpha^4) \\ g(X) &= \alpha^3 + \alpha X + X^2 + \alpha^3 X^3 + X^4 \end{aligned}$$

Dále si přepíšeme zprávu k přenesení do polynomiálního tvaru:

$$i(X) = \alpha + \alpha^3 X + \alpha^5 X^2$$

Nyní můžeme pokračovat podle vztahu 1.13:

$$
\begin{aligned}
c(X) &= X^{n-k} \cdot i(X) + p(X) = X^{n-k} \cdot i(X) + Rem\left\{\frac{X^{(n-k)} \cdot i(X))}{g(X)}\right\} \\
X^{n-k} \cdot i(X) &= X^4 \cdot i(X) = \alpha X^4 + \alpha^3 X^5 + \alpha^5 X^6 \\
Rem\left\{\frac{X^{(n-k)} \cdot i(X))}{g(X)}\right\} &= \alpha^6 X^3 + \alpha^4 X^2 + \alpha^2 X + \alpha^0
\end{aligned}
$$

Dosazením dílčích výsledků dostáváme zakódovanou zprávu:

$$
c(X) = X^{n-k} \cdot i(X) + p(X) = \alpha^5 X^6 + \alpha^3 X^5 + \alpha X^4 + \alpha^6 X^3 + \alpha^4 X^2 + \alpha^2 X + \alpha^0
$$

Převedením z polynomiálního tvaru tedy dostaneme zprávu
$c = (111\ \ 110\ \ 010\ \ 101\ \ 011\ \ 001\ \ 100)$.

2 DEKÓDOVÁNÍ RS KÓDŮ

Pro dekódování RS kódů bylo vyvinuto mnoho algoritmů. V této kapitole se pokusím vysvětlit obecný postup dekódování RS kódů. V následující kapitole pak budou podrobněji popsány jednotlivé možnosti přístupu k dekódování.

Obecně lze podle [12] algebraické dekódování RS kódů rozdělit do následujících kroků:

1. Výpočet syndromů (určení zda nastala či nenastala chyba).

2. Stanovení chybového polynomu (lokátoru chyb), jehož kořeny nám určují, kde se nachází chyby v přenášeném kódovém slovu. Tento krok lze řešit více způsoby, existuje několik odlišných algoritmů pro nalezení chybového polynomu, např. Peterson-Gorenstein-Zierleův algoritmus, Berlekamp-Maseyův algoritmus, Euklidův algoritmus atd.

3. Nalezení kořenů chybového polynomu. To se obvykle provádí pomocí Chienova vyhledávání.

4. Stanovení hodnoty chyby. To je obvykle dosaženo použitím Forneyova algoritmu.

5. Oprava chyby.

2.1 Výpočet syndromů

Prvním krokem dekódování RS kódů je výpočet syndromů. Jestliže:

$$g(\alpha) = g(\alpha^2) = \ldots = g(\alpha^{2t}) = 0, \tag{2.1}$$

vyplývá, že kódové slovo $\mathbf{c} = (c_0, \ldots, c_{n-1})$ s polynomem $c(X) = c_o + \ldots + c_{n-1}X^{n-1}$ bude podle [16]:

$$c(\alpha) = \ldots = c(\alpha^{2t}) = 0. \tag{2.2}$$

Za předpokladu přijatého polynomu ve tvaru $r(X) = c(X) + e(X)$ pak můžeme psát:

$$s_j = r(\alpha^j) = e(\alpha^j) = \sum_{k=0}^{n-1} e_k \alpha^{jk}, \qquad j = 1, 2, \ldots, 2t. \tag{2.3}$$

Hodnoty $s_1, s_2, \ldots, s_{2t}$ jsou nazývány syndromy přijatých dat.

Předpokládejme, že $\mathbf{r}$ obsahuje υ chyb, které se nachází v lokacích $i_1, i_2, \ldots, i_\upsilon$, s odpovídající chybovou hodnotou v těchto lokacích $e_{ij} \neq 0$, potom [16]:

$$s_j = \sum_{l=1}^{\upsilon} e_{il} (^j)^{i_l} = \sum_{l=1}^{\upsilon} e_{il} (^{i_l})^j. \tag{2.4}$$

Nechť

$$X_l = \alpha^{i_l}. \tag{2.5}$$

Pak můžeme psát podle [16]:

$$s_j = \sum_{l=1}^{\upsilon} e_{il} X_l^j, \qquad j = 1, 2, , \ldots, 2t. \tag{2.6}$$

Jestliže známe X_l, potom známe pozici dané chyby. Například řekněme, že známe $X_1 = \alpha^4$. To znamená, podle definice X_l, že $i_1 = 4$; to znamená, že chyba je v přijaté číslici r_4. X_l tedy můžeme podle [16] nazvat lokátor chyby.

2.2 Lokalizační mnohočlen

V tomto kroku se liší jednotlivé přístupy k dekódování RS kódů. V dalším textu jsou popsány jednotlivé metody a s nimi i jednotlivé algoritmy pro nalezení lokalizačního mnohočlenu.

Obecně je pak lokalizační mnohočlen podle [12] definován jako:

$$L(X) = \prod_{l=1}^{\upsilon} (1 - X_l x) = L_\upsilon X^\upsilon + L_{\upsilon-1} X^{\upsilon-1} + \ldots + L_1 X + L_0, \tag{2.7}$$

kde $L_0 = 1$. Ostatní koeficienty L_υ je nutné dopočítat některým z algoritmů uvedených v kapitole 3.1 až 3.3.

2.3 Chienovo vyhledávání

Předpokládejme, že v tuto chvíli máme stanovený lokalizační mnohočlen. Dalším krokem je nalezení kořenů tohoto mnohočlenu. To spočívá v tom, že se postupně prověřují všechny prvky z Galoisova tělesa, jestli nejsou kořenem mnohočlenu. Existují i odlišné způsoby [1] [6], ale pro tělesa používané pro korekční kódy a pro počet hledaných kořenů se jeví Chienovo vyhledávání jako nejvíc účinné [12].

Vezměme příklad, kdy máme $\upsilon = 3$ a lokalizační mnohočlen je tedy podle vztahu 2.7 ve tvaru:

$$L(X) = L_0 + L_1X + L_2X^2 + L_3X^3 = 1 + L_1X + L_2X^2 + L_3X^3 \tag{2.8}$$

Vypočítáme $L(X)$ pro každý nenulový prvek Galoisova tělesa: $X = 1, X = \alpha$, $X = \alpha^2, \ldots, X = \alpha^{q^m-2}$. To nám dává následující:

$$\begin{aligned}
L(1) &= 1 + L_1(1) + L_2(1)^2 + L_3(1)^3 \\
L(\alpha) &= 1 + L_1(\alpha) + L_2(\alpha)^2 + L_3(\alpha)^3 \\
L(\alpha^2) &= 1 + L_1(\alpha^2) + L_2(\alpha^2)^2 + L_3(\alpha^2)^3 \\
&\vdots \\
L(\alpha^{q^m-2}) &= 1 + L_1(\alpha^{q^m-2}) + L_2(\alpha^{q^m-2})^2 + L_3(\alpha^{q^m-2})^3
\end{aligned}$$

V případě, že $L(X)$ vyjde nulové pro daný prvek GF, je indikována pozice chyby. Jestliže nejsou nalezeny žádné kořeny, znamená to neodstranitelné chyby.

2.4 Výpočet hodnoty chyb (Forneyův algoritmus)

Po získání lokátoru chyb a získání jeho kořenů nám zbývá vyčíslit hodnotu chyb. Pro toto byl vyvinut Forneyův algoritmus, pro jeho odvození se musíme vrátit k výpočtu syndromů.

V případě, že máme určené syndromy $S_1, S_2, \ldots, S_{2t}$ můžeme určit syndromový polynom $S(X)$ následovně:

$$S(X) = S_1 + S_2(X) + \ldots + S_{2t}X^{2t-1} = \sum_{j=0}^{2t-1} S_{j+1}X^j. \tag{2.9}$$

Následně můžeme podle [12] definovat polynom pro určení hodnoty chyb $E(X)$:

$$E(X) = S(X) \cdot L(X)(modX^{2t}) \tag{2.10}$$

Tato rovnice je nazývána klíčová rovnice (ang. key equation). Operace $modX^{2t}$ zajišťuje zrušení všech členů polynomu, které jsou stupně $2t$ nebo vyšší. Protože již máme určený polynom pro určení chyb, stačí vypočítat derivaci $L(X)$ a můžeme podle [8] a [12] psát vztah pro hodnotu chyby M_l, $l \in 1, \ldots, \upsilon$ která je obecně ve tvaru:

$$M_l = -\frac{E(P_l^{-1})}{L'(P_l^{-1})} \tag{2.11}$$

kde $L'(X)$ je derivace lokalizačního mnohočlenu.

2.5 Oprava chyby

V poslední fázi jsou již známy pozice chyb a jejich hodnoty v přenášené zprávě. Na základě těchto hodnot je sestrojen chybový polynom $e(X)$. Tento polynom je přičítán k polynomu, který byl přijatý v dekodéru $r(X)$, jejich sečtením pak dostáváme opravenou vstupní zprávu $i(X)$. Toto vyjadřuje následující rovnice:

$$i(X) = r(X) + e(X). \tag{2.12}$$

3 ALGORITMY PRO STANOVENÍ CHYBOVÉHO MNOHOČLENU

3.1 Berlekamp - Masseyův algoritmus

V této kapitole bude popsán Berlekamp - Masseyův algoritmus (dále jen BM algoritmus) pro stanovení chybového polynomu. Tento algoritmus je založen na postupné iteraci.

BM algoritmus můžeme formálně shrnout Blahutovou interpretací[8], která je vidět na vývojovém diagramu na obr. 3.1 . Jednotlivé fáze algoritmu jsou znázorněny v diagramu a tyto kroky jsou popsány níže [8]:

1. Nastavení počátečních podmínek:

   index iterace: $i = 0$

   délka LSFR: $l^{(1)} = 0$

   chybový mnohočlen: $L^{(1)}(X) = 0$

   pomocný chybový mnohočlen: $A^{(1)}(X) = 0$
2. Kontrola, zda jsme dosáhli konce iterace, to je jestliže $i = 2t$.
3. Odhad chyby, která náleží k dalšímu vygenerovanému syndromu:

$$d^i = \sum_{n=0}^{l^i} L_n^{(i)}.s_{i+1-n} \tag{3.1}$$

4. Kontrola, zda je odchylka $d^i = 0$ a v případě, že ano, pak aktuální LFSR o délce l^i a chybový mnohočlen $L^i(X)$ produkuje následující syndrom s_{i+1}. Proto přejděte ke kroku 5, v opačném případě přejděte ke kroku 6.
5. Jednoduše posuň pomocný LFSR o jednu pozici pomocí následujícího vztahu:

$$A^{(i)}(X) := X.A^{(i)}(X), \tag{3.2}$$

   dále pokračuj krokem 12.
6. Jestliže $d^{(i)} \neq 0$, opravíme dočasný chybový mnohočlen $T^{(i)}(X)$. Toto vyjadřuje následující vztah:

$$T^{(i)}(X) = L^{(i)}(X) - d^{(i)} - d^{(i)}.X.A^{(i)}(X). \tag{3.3}$$

7. Ověření, jestli je LFSR třeba zvětšit, nebo ne. V případě že ne, pokračujeme krokem 8, jinak krokem 9.
8. Obnovení chybového mnohočlenu pomocí následujícího vztahu:

$$L^{(i)}(X) := T^{(i)}(X) \tag{3.4}$$

   vracíme se do kroku 5.
9. Normalizujeme poslední chybový mnohočlen $L(X)$ dělením $d^{(i)} \neq 0$ a výsledek se uloží do pomocného LFSR $A^{(i)}(X)$:

$$A^{(i)}(X) := \frac{L^{(i)}(X)}{d^{(i)}} \tag{3.5}$$

   pokračujeme krokem 10.
10. Nyní můžeme přepsat $L^{(i)}(X)$, protože byl v předchozím kroku normalizován a uložen do $A^{(i)}(X)$. $L^{(i)}(X)$ je aktualizován následovně:

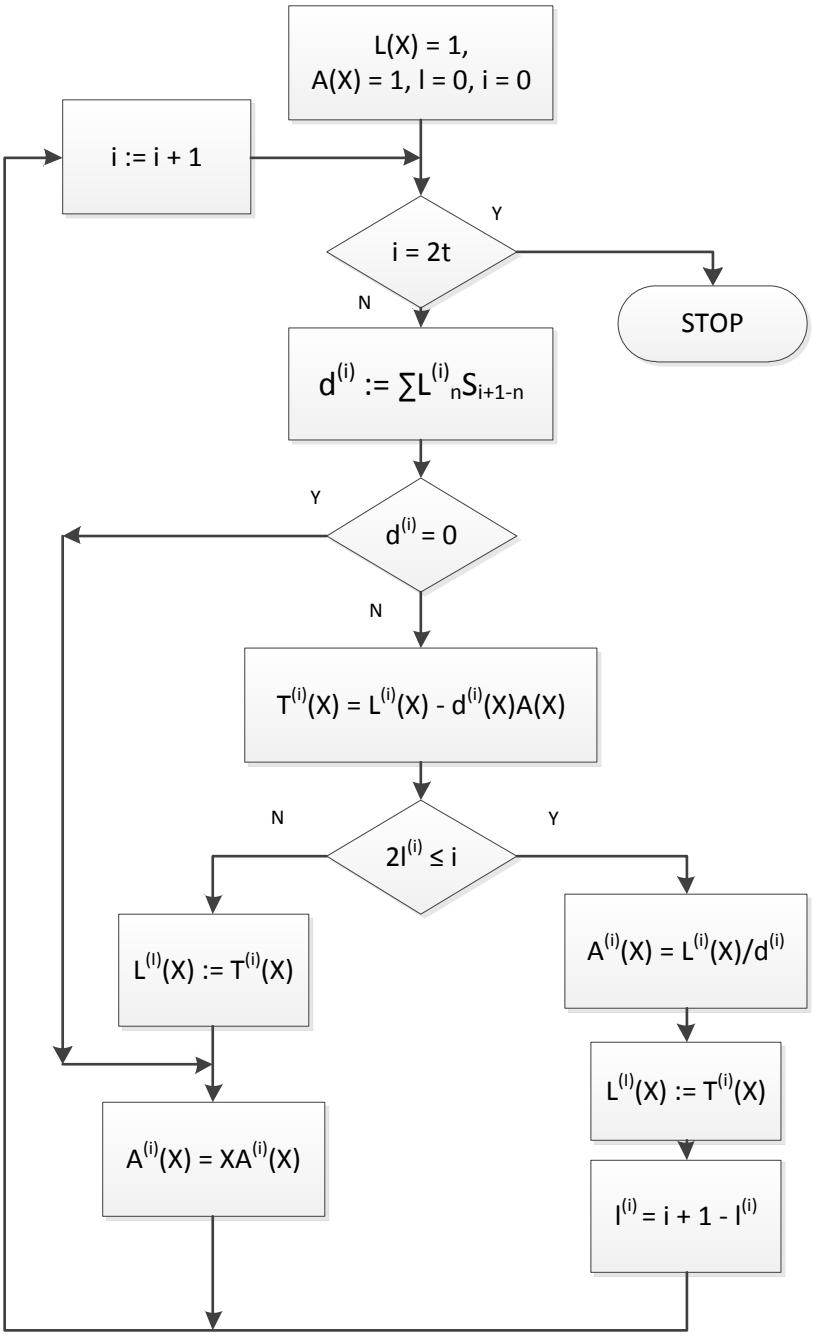
$$L^{(i)}(X) := T^{(i)}(X) \tag{3.6}$$

   pokračujeme krokem 11.
11. Podle kroku 7 musí být prodloužen LFSR. To provedeme pomocí následujícího vztahu:

$$l^{i+1} = i + 1 - l^{i}. \tag{3.7}$$

   pokračujeme krokem 12.
12. Inkrementujeme index iterace i a vrátíme se do kroku 2.

Pro podrobnější studium tohoto algoritmu bych doporučil literaturu [12] [8] [5] a [13].



Obr. 3.1: Vývojový diagram pro Berlekamp - Masseyův algoritmus

3.1.1 Příklad dekódování BM algoritmem

Pro upřesnění výkladu Berlekamp-Masseyova algoritmu uvedeme příklad. Máme přijatou zprávu $r = (000\ 111\ 000\ 101\ 000\ 000\ 000)$. Posloupnost byla zakódována kódem RS (7, 3), který je definován polem GF (2^3) nad základním polynomem $p_i(X) = 1 + X^2 + X^3$ viz tabulka 3.1.

Exponenciální tvar	Polynomiální tvar	Binární tvar
0	0	0 0 0
α^0	1	1 0 0
α	α	0 1 0
α^2	α^2	0 0 1
α^3	$1 + \alpha^2$	1 0 1
α^4	$1 + \alpha + \alpha^2$	1 1 1
α^5	$1 + \alpha$	1 1 0
α^6	$\alpha + \alpha^2$	0 1 1

Tab. 3.1: Galoisovo pole generované polynomem $p_i(X) = 1 + X^2 + X^3$

Nejprve je potřeba přepsat přijatou zprávu do polynomiálního tvaru, který je nutný pro výpočet syndromů:

$$r(X) = \alpha^4 X + \alpha^3 X^3$$

Následuje samotný výpočet syndromů:

$$\begin{aligned}
s_1(\alpha) &= r(\alpha) = \alpha^5 + \alpha^6 = \alpha^3 \\
s_2(\alpha) &= r(\alpha^2) = \alpha^6 + \alpha^2 = \alpha \\
s_3(\alpha) &= r(\alpha^3) = 1 + \alpha^5 = \alpha \\
s_4(\alpha) &= r(\alpha^4) = \alpha + \alpha = 0
\end{aligned}$$

Dále budeme postupovat podle algoritmu uvedeného v kapitole 3.1. Začneme nastavením počátečních podmínek:

$$i = 0, \quad l = 0, \quad A(X) = 1, \quad L(X) = L_0 = 1$$

Otestujeme, zda se nedosáhlo konce iterace, to je když $i = 2t$, $0 \neq 4 \Rightarrow$ konec iterace nenastal. Můžeme pokračovat krokem 3, to je odhad chyby:

$$d = \sum_{n=0}^{l} L_n \cdot S_{i+1-n} = L_0 \cdot S_1 = 1 \cdot \alpha^3 = \alpha^3$$

Následuje kontrola, zda je $d = 0$. Protože je $d = \alpha^3 \neq 0$ je potřeba aktualizovat dočasný polynom $T(X)$ a proto pokračujeme krokem 6.

$$T(X) = L(X) - d \cdot X \cdot A(X) = 1 - \alpha^3 X$$

Ověření, zda musí být zvětšen LSFR, $2l \leq i$ platí a tedy musíme LSFR zvětšit $\Rightarrow$ krok 9.

$$A(X) = \frac{L(X)}{d} = \frac{1}{\alpha^3} = \alpha^4$$

Nyní je třeba aktualizovat dočasný lokalizační mnohočlen:

$$L(X) = T(X) = 1 - \alpha^3 X$$

Pokračujeme krokem 11, což znamená prodloužení LFSR:

$$l = i + 1 - l = 0 + 1 - 0 = 1$$

Tímto je dokončena první iterace algoritmu, inkrementujeme index iterace $i := i + 1$ a pro přehlednost uvedeme aktuální hodnoty a vracíme se ke kroku 2.

$$i = 1, \quad l = 1, \quad A(X) = \alpha^4, \quad L(X) = L_0 = 1 - \alpha^3 X$$

Otestujeme, zda se nedosáhlo konce iterace, to je když $i = 2t$, $1 \neq 4 \Rightarrow$ konec iterace nenastal. Můžeme pokračovat krokem 3:

$$d = \sum_{n=0}^{l} L_n \cdot S_{i+1-n} = L_0 \cdot S_2 + L_1 \cdot S_1 = 1 \cdot \alpha + (-\alpha^3) \cdot \alpha^3 = \alpha^2$$

Protože $d \neq 0$ pokračujeme krokem 6:

$$T(X) = L(X) - d \cdot X \cdot A(X) = 1 - \alpha^3 X - \alpha^2 X \alpha^4 = 1 + \alpha^5$$

$2l \leq i$ neplatí, proto můžeme přeskočit na krok 8 dále pak krok 5:

$$L(X) = T(X) = 1 + \alpha^5$$

$$A(X) = A(X) \cdot X = \alpha^4 X$$

Tím jsme na konci druhé iterace algoritmu a proto opět inkrementujeme index iterace $i := i + 1$ a pro přehlednost znovu uvedeme aktuální hodnoty a vracíme se ke kroku 2.

$$i = 2, \quad l = 1, \quad A(X) = \alpha^4 X, \quad L(X) = 1 + \alpha^5 X$$

Otestujeme, zda se nedosáhlo konce iterace, to je když $i = 2t$, $2 \neq 4 \Rightarrow$ konec iterace nenastal. Můžeme pokračovat krokem 3:

$$d = \sum_{n=0}^{l} L_n \cdot S_{i+1-n} = L_0 \cdot S_3 + L_1 \cdot S_2 = 1 \cdot \alpha + \alpha^5 \cdot \alpha = \alpha^2$$

Protože $d \neq 0$ pokračujeme krokem 6:

$$T(X) = L(X) - d \cdot X \cdot A(X) = 1 + \alpha^5 X - \alpha^2 X \alpha^4 X = 1 + \alpha^5 X + \alpha^6 X^2$$

$2l \leq i$ platí, a proto pokračujeme krokem 9, 10 a 11:

$$A(X) = \frac{L(X)}{d} = \frac{1+\alpha^5 X}{\alpha^2} = \alpha^5 + \alpha^3 X$$

$$L(X) = T(X) = 1 + \alpha^5 X + \alpha^6 X^2$$

$$l = i + 1 - l = 2$$

Následuje další iterace: $i := i + 1$, aktuální hodnoty nyní jsou:

$$i = 3, \quad l = 2, \quad A(X) = \alpha^5 + \alpha^3 X, \quad L(X) = 1 + \alpha^5 X + \alpha^6 X^2$$

Otestujeme,zda se nedosáhlo konce iterace, to je když $i = 2t$, $2 \neq 4 \Rightarrow$ konec iterace nenastal. Můžeme pokračovat krokem 3:

$$d = \sum_{n=0}^{l} L_n \cdot S_{i+1-n} = L_0 \cdot S_4 + L_1 \cdot S_3 + L_2 \cdot S_2 = 1 \cdot 0 + \alpha^5 \cdot \alpha + \alpha^6 \cdot \alpha = \alpha^4$$

Protože $d \neq 0$ pokračujeme krokem 6:

$$T(X) = L(X) - d \cdot X \cdot A(X) = 1 + \alpha^5 X + \alpha^6 X^2 - \alpha^4 X \cdot (\alpha^5 + \alpha^3 X) = 1 + \alpha^4 X + \alpha^4 X^2$$

$2l \leq i$ neplatí, proto můžeme přeskočit na krok 8 dále pak krok 5:

$$L(X) = T(X) = 1 + \alpha^4 X + \alpha^4 X^2$$

$$A(X) = X \cdot A(X) = \alpha^5 X + \alpha^3 X^2$$

Dospěli jsme ke konci další iterace, zvýšíme index inkrementace: $i := i + 1 = 4 = 2t$ a jelikož $i = 2t$, další iterace už neproběhne a dosáhli jsme konečného lokalizačního mnohočlenu $L(X) = 1 + \alpha^4 X + \alpha^4 X^2$. Nyní můžeme pomocí Chienova vyhledávání nalézt pozice chyb:

$$
\begin{aligned}
L(\alpha^0) &= \alpha^4 + \alpha^4 + 1 = 1 \\
L(\alpha) &= \alpha^6 + \alpha^5 + 1 = \alpha^2 \\
L(\alpha^2) &= \alpha + \alpha^6 + 1 = \alpha^3 \\
L(\alpha^3) &= \alpha^3 + \alpha^0 + 1 = \alpha^3 \\
L(\alpha^4) &= \alpha^5 + \alpha + 1 = 0 \leftarrow \\
L(\alpha^5) &= \alpha^0 + \alpha^2 + 1 = \alpha^2 \\
L(\alpha^6) &= \alpha^2 + \alpha^3 + 1 = 0 \leftarrow
\end{aligned}
$$

Nalezené pozice musíme invertovat:

$$
\begin{aligned}
(\alpha^4)^{-1} &= \frac{\alpha^0}{\alpha^4} = \alpha^3 = P_1 \\
(\alpha^6)^{-1} &= \frac{\alpha^0}{\alpha^6} = \alpha^1 = P_2
\end{aligned}
$$

Nyní je třeba vyjádřit hodnotu chyb na daných pozicích, to provedeme pomocí Forneyova algoritmu. Nejprve je nutno vypočítat polynom pro odhad chyby $E(X)$.

$$
\begin{aligned}
E(X) &= L(X) \cdot S(X)(modX^4) \\
E(X) &= (1 + \alpha^4 X + \alpha^4 X^2) \cdot (\alpha^3 X + \alpha X^2 + \alpha X^3)(modX^4) \\
E(X) &= \alpha^5 X^2 + \alpha^3 X
\end{aligned}
$$

Nyní již máme všechno potřebné pro výpočet hodnoty chyby a můžeme vypočítat hodnoty chyb:

$$
\begin{aligned}
M_l &= \frac{E(P_l^{-1})}{\prod_{j \neq l}(1 - P_j P_l^{-1})} \\
M_1 &= \frac{E(P_1^{-1})}{(1 - P_2 P_1^{-1})} = \frac{\alpha^5 \cdot (\alpha^4)^2 + \alpha^3 \cdot \alpha^4}{1 - \alpha \cdot \alpha^4} = \alpha^3 \\
M_2 &= \frac{E(P_2^{-1})}{(1 - P_1 P_2^{-1})} = \frac{\alpha^5 \cdot \alpha^1 + \alpha^3 \cdot \alpha^6}{1 - \alpha^3 \cdot \alpha^6} = \alpha^4
\end{aligned}
$$

Nalezením pozic chyb a vypočtením jejich hodnot jsme získali chybový polynom $e(X) = \alpha^4 X + \alpha^3 X^3$, jeho přičtením k přijatému polynomu $r(X)$ získáme nulový vektor, což je hledaná vstupní zpráva kodéru.

3.2 Peterson-Gorenstein-Zierleův algoritmus

Peterson-Gorenstein-Zierleova (dále jen PGZ) metoda dekódování obsahuje inverzi dvou matic. Jednu pro výpočet chybového polynomu a druhou pro určení hodnot chyb.

Následující výpočty vycházejí ze vztahu podle [8]:

$$\mathbf{L} = \mathbf{S}^{-1} \cdot \mathbf{S}. \tag{3.8}$$

Řešení je založeno na následující teorému. *Vandermondova* matice $\mathbf{S}$ tvořená syndromy není singulární a může být invertována, pokud je její dimenze $v \times v$, pokud je singulární, tak nemůže být invertována protože dimenze je větší než v, kde v je aktuální počet chyb, který nastal.

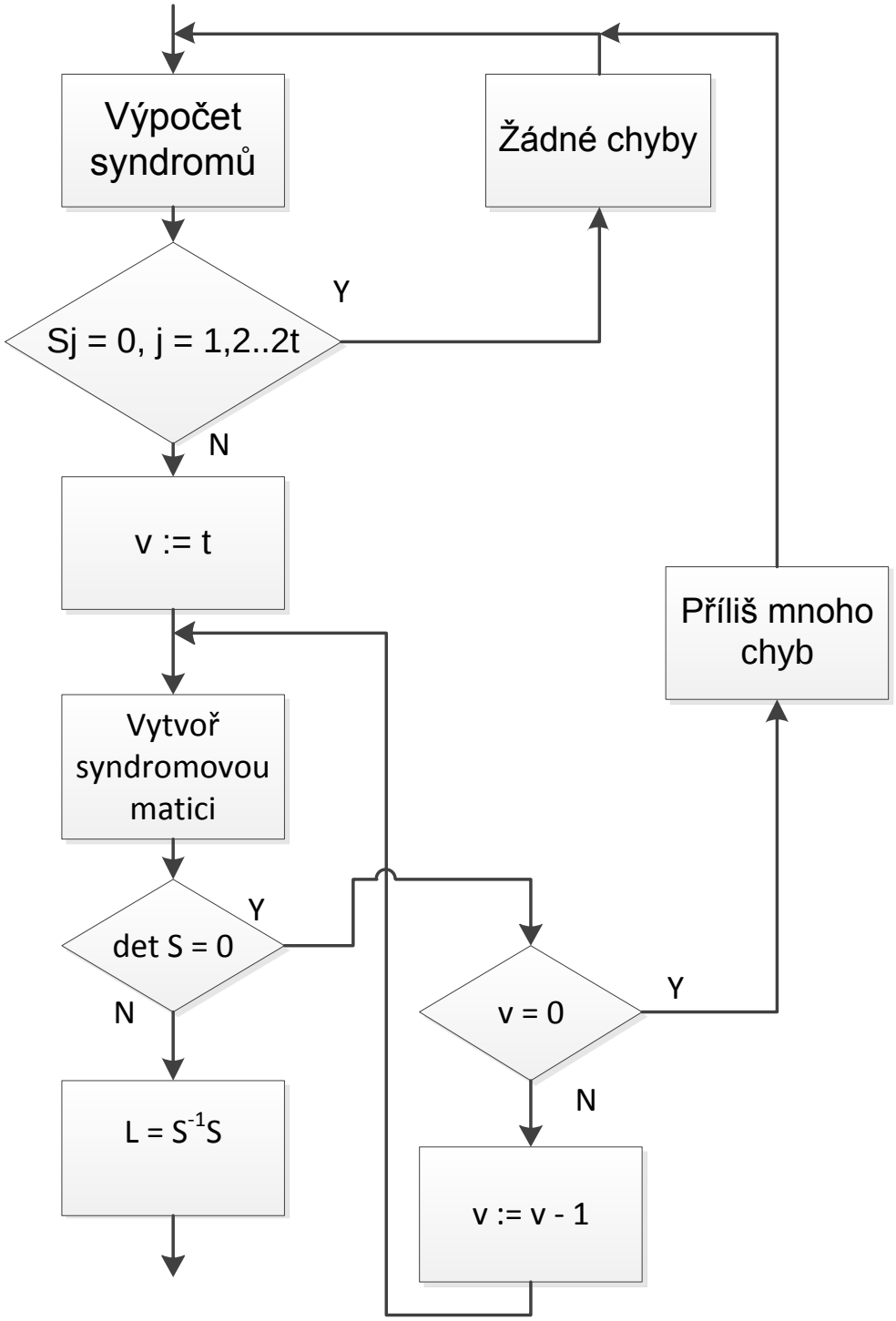
Abychom mohli vyřešit rovnici 3.8, musíme určit v pro schopnost invertace $\mathbf{S}$. Zpočátku nastavíme $v = t$, protože t je maximální možný počet chyb, a počítáme determinant $\mathbf{S}$. Pokud je $det(\mathbf{S}) = 0$, při $v = t$ pak musíme dekrementovat v o jedna, postupně až na nulu, dokud nenajdeme v, pro které bude $det(\mathbf{S}) \neq 0$. Jakmile je nalezeno správné v, spočítáme $\mathbf{S}^{-1}$ a určíme $L = \mathbf{S}^{-1}\mathbf{S}$.

Následuje využití Chienova vyhledávání. Jsou nalezeny pozice chyb a zbývá pouze vyčíslit hodnotu chyby. To lze provést výše zmíněným Forneynovým algoritmem, nicméně v době vyvinutí PGZ metody tento algoritmus nebyl znám, a tak se tento problém řešil opět pomocí invertování matice.

Je to poměrně jednoduché, vyjdeme ze vztahu $\mathbf{M} = \mathbf{P}^{-1}\mathbf{S}$ podle [8]. Z Chienova vyhledávání již známou matici pozic chyb $\mathbf{P}$ invertujeme pro spočítání vektoru velikosti chyb $\mathbf{M}$:

$$\mathbf{M} = \mathbf{P}^{-1}\mathbf{S}. \tag{3.9}$$

Celá metoda PGZ je ukázána na vývojovém diagramu na obr. 4.3.



Obr. 3.2: Vývojový diagram pro Peterson-Gorenstein-Zierler dekódování

3.2.1 Příklad dekódování PGZ algoritmem

Stejně jako u předchozí metody pro lepší pochopení opět uvedeme příklad. Použijeme stejné zadání jako u předchozí metody. Máme přijatou zprávu $r = (000$ $111\ 000\ 101\ 000\ 000\ 000)$. Posloupnost byla zakódována kódem RS (7, 3), který je definován polem GF (2^3) nad základním polynomem $p_i(X) = 1 + X^2 + X^3$ viz tabulka 3.1.

Přijatou zprávu si přepíšeme do polynomiálního tvaru:

$$r(X) = \alpha^4 X + \alpha^3 X^3$$

Následuje výpočet syndromů:

$$\begin{aligned} s_1(\alpha) &= r(\alpha) = \alpha^5 + \alpha^6 = \alpha^3 \\ s_2(\alpha) &= r(\alpha^2) = \alpha^6 + \alpha^2 = \alpha \\ s_3(\alpha) &= r(\alpha^3) = 1 + \alpha^5 = \alpha \\ s_4(\alpha) &= r(\alpha^4) = \alpha + \alpha = 0 \end{aligned}$$

Vzhledem k tomu, že nemáme žádné informace o skutečném počtu chyb v přijímači, musíme nejprve očekávaný počet chyb ν určit. Zpočátku nastavíme $\nu = t = 2$. Otestujeme, zda je determinant matice $\nu \times \nu$ různý od nuly:

$$det(\mathbf{S}) = det \begin{vmatrix} s_1 & s_2 \\ s_2 & s_3 \end{vmatrix} = (s_1 s_3 - s_2^2)$$

Dosazením za jednotlivé syndromy dostaneme:

$$det(\mathbf{S}) = (\alpha^3 \alpha - \alpha^2) = \alpha^4 - \alpha^2 = \alpha^5 \neq 0$$

Protože je $det(\mathbf{S}) \neq 0$ můžeme vypočítat $\mathbf{S}^{-1}$:

$$\left[\begin{array}{cc|cc} \alpha^3 & \alpha & \alpha^0 & 0 \\ \alpha & \alpha & 0 & \alpha^0 \end{array} \right]$$

$$\left[\begin{array}{cc|cc} \alpha^3 & \alpha & \alpha^0 & 0 \\ 0 & \alpha^2 & \alpha^5 & \alpha^0 \end{array}\right]$$

$$\left[\begin{array}{cc|cc} \alpha^3 & 0 & \alpha^6 & \alpha^6 \\ 0 & \alpha^2 & \alpha^5 & \alpha^0 \end{array}\right]$$

$$\left[\begin{array}{cc|cc} \alpha^0 & 0 & \alpha^3 & \alpha^3 \\ 0 & \alpha^0 & \alpha^3 & \alpha^5 \end{array}\right]$$

Výsledná $\mathbf{S}^{-1}$ je tedy:

$$\mathbf{S}^{-1} = \begin{bmatrix} \alpha^3 & \alpha^3 \\ \alpha^3 & \alpha^5 \end{bmatrix}$$

Nyní už můžeme vypočítat hledaný lokalizační mnohočlen:

$$\begin{bmatrix} L_1 \\ L_2 \end{bmatrix} = \begin{bmatrix} \alpha^3 & \alpha^3 \\ \alpha^3 & \alpha^5 \end{bmatrix} \cdot \begin{bmatrix} \alpha \\ 0 \end{bmatrix} = \begin{bmatrix} \alpha^4 \\ \alpha^4 \end{bmatrix}$$

Lokalizační mnohočlen je tedy $L(X) = \alpha^4 X^2 + \alpha^4 X + 1$ a pomocí Chienova vyhledávání můžeme najít pozice chyb:

$$\begin{aligned}
L(\alpha^0) &= \alpha^4 + \alpha^4 + 1 = 1 \\
L(\alpha) &= \alpha^6 + \alpha^5 + 1 = \alpha^2 \\
L(\alpha^2) &= \alpha + \alpha^6 + 1 = \alpha^3 \\
L(\alpha^3) &= \alpha^3 + \alpha^0 + 1 = \alpha^3 \\
L(\alpha^4) &= \alpha^5 + \alpha + 1 = 0 \leftarrow \\
L(\alpha^5) &= \alpha^0 + \alpha^2 + 1 = \alpha^2 \\
L(\alpha^6) &= \alpha^2 + \alpha^3 + 1 = 0 \leftarrow
\end{aligned}$$

Nalezené pozice je potřeba invertovat:

$$(\alpha^4)^{-1} = \frac{\alpha^0}{\alpha^4} = \alpha^3 = P_1$$
$$(\alpha^6)^{-1} = \frac{\alpha^0}{\alpha^6} = \alpha^1 = P_2$$

Nyní už zbývá pouze vypočítat hodnoty chyb na daných pozicích. V našem případě, kdy $\nu = 2$ je výpočet jednoduchý, dostáváme dvě rovnice o dvou neznámých:

$$\begin{aligned}
s_1 &= M_1 P_1 + M_2 P_2 \\
s_2 &= M_1 P_1^2 + M_2 P_2^2 \\
\alpha^3 &= M_1 \alpha^3 + M_2 \alpha \\
\alpha &= M_1 \alpha^6 + M_2 \alpha^2
\end{aligned}$$

$$\begin{aligned}
M_1 &= \frac{\alpha^3 + M_2\alpha}{\alpha^3} = (\alpha^3 + M_2\alpha) \cdot \alpha^4 = \alpha^0 + M_2\alpha^5 \\
\alpha &= (\alpha^0 + M_2\alpha^5) \cdot \alpha^6 + M_2\alpha^2 \\
\alpha &= \alpha^6 + M_2\alpha^4 + M_2\alpha^2 \\
\alpha - \alpha^6 &= M_2 \cdot (\alpha^4 + \alpha^2) \\
M_2 &= \frac{\alpha + \alpha^6}{\alpha^4 + \alpha^2} = \frac{\alpha^2}{\alpha^5} = \alpha^4 \\
M_1 &= \alpha^0 + M_2\alpha^5 = \alpha^0 + \alpha^2 = \alpha^3
\end{aligned}$$

Nalezením pozic chyb a výpočtem jejich velikosti jsme dostali chybový polynom $e(X) = \alpha^4 X + \alpha^3 X^3$ a můžeme tedy opravit zprávu přijatou do dekodéru. Sečtením $e(X)$ a $r(X)$ vidíme, že vstupní slovo kodéru byl nulový vektor.

3.3 Euklidův algoritmus

V této kapitole si ukážeme použití Euklidova algoritmu pro konstrukci chybového polynomu. Tento přístup k dekódování se často nazývá Sugiyama algoritmus.

Vyjdeme ze základní rovnice podle [12]:

$$E(X) = S(X)L(X) \quad (mod X^{2t}) \tag{3.10}$$

Známe pouze $S(X)$ a t a chceme určit chybový polynom $L(X)$ a polynom pro určení velikosti chyb $E(X)$. Tento problém se jeví jako neřešitelný. Nicméně rovnici 3.8 můžeme podle [12] přepsat na:

$$\Theta(X)(X^{2t}) + L(X)S(X) = E(X) \tag{3.11}$$

pro libovolný polynom $\Theta(X)$. Uveďme, že podle [12] rozšířený Euklidův algoritmus vrací pro dvojici prvků (a, b), dvojici prvků (s, t) takových, že:

$$as + bt = c \tag{3.12}$$

kde c je největší společný dělitel z a a b. Z této úvahy vychází Euklidova metoda a celý proces lze pak podle [5] shrnout do následujících kroků:

1. Spočítání syndromů a syndromového polynomu $S(X) = s_1 + s_2X + \ldots + s_{2t}X^{2t-1}$.
2. Jestliže $S(X) = 0$, pak je odpovídající přijatý vektor považován za kódový vektor.
3. Jestliže $S(X) \neq 0$, pak je algoritmus inicializován jako:

$$\begin{aligned} r_{-1}(X) &= X^{2t} \\ r_0 &= S(X) \\ t_{-1}(X) &= 0 \\ t_0(X) &= 1 \\ i &= -1 \end{aligned} \tag{3.13}$$

4. Rekurzivní parametry jsou stanoveny následovně:

$$r_i(X) = r_{i-2}(X) - q(X)r_{i-1}(X) \tag{3.14}$$

$$t_i(X) = t_{i-2}(X) - q(X)t_{i-1}(X) \tag{3.15}$$

   tato rekurze probíhá tak dlouho dokud je $(r_i(X)) \geq t_i$.

5. Když $(r_i(X)) < t_i$, rekurze skončí a můžeme určit:

$$E(X) = r_i(X) \tag{3.16}$$

$$L(X) = t_i(X) \tag{3.17}$$

6. Následuje využití Chienova vyhledávání a Forneyův algoritus, viz kapitola 2.3 a 2.4.

Pro podrobnější studium Euklidova algoritmu bych doporučil literaturu [12] a [5].

3.3.1 Příklad dekódování Euklidovým algoritmem

Pro lepší pochopení Euklidova algoritmu uvedeme příklad. Máme přijatou zprávu $r = (000\ 111\ 000\ 101\ 000\ 000\ 000)$. Posloupnost byla zakódována kódem RS $(7, 3)$, který je definován polem GF (2^3) nad základním polynomem $p_i(X) = 1 + X^2 + X^3$ viz tabulka 3.1.

Nejprve je nutné přepsat přijatou zprávu do polynomiálního tvaru, který je nutný pro výpočet syndromů:

$$r(X) = \alpha^4 X + \alpha^3 X^3$$

Následuje samotný výpočet syndromů:

$$\begin{aligned} s_1(\alpha) &= r(\alpha) = \alpha^5 + \alpha^6 = \alpha^3 \\ s_2(\alpha) &= r(\alpha^2) = \alpha^6 + \alpha^2 = \alpha \\ s_3(\alpha) &= r(\alpha^3) = 1 + \alpha^5 = \alpha \\ s_4(\alpha) &= r(\alpha^4) = \alpha + \alpha = 0 \end{aligned}$$

$$S(X) = \alpha^3 + \alpha X + \alpha X^2$$

i	$r_i = r_{i-2} - q_i r_{i-1}$	q_i	$t_i = t_{i-2} - q_i t_{i-1}$
-1	$X^{n-k} = X^4$		0
0	$S(X) = \alpha^3 + \alpha X + \alpha X^2$		1
1	$X + \alpha^5$	$\alpha^2 + \alpha^6 X + \alpha^4 X^2$	$\alpha^2 + \alpha^6 X + \alpha^4 X^2$

Tab. 3.2: Tabulka znázorňující iterace Euklidova algoritmu.

Jednotlivé kroky Euklidova algoritmu jsou v tabulce 3.2. Jestliže je stupeň polynomu ve sloupci r_i menší než stupeň polynomu ve sloupci t_i, algoritmus se ukončí. Potom můžeme psát:

$$\begin{aligned} E_1(X) &= X + \alpha^5 \\ L_1(X) &= \alpha^6 X^2 + \alpha^6 X + \alpha^2 \end{aligned}$$

Polynom $L_1(X)$ takto získaný, je násobený prvkem Galoisova pole $\lambda \in GF(2^3)$ za cílem převést jej do normovaného polynomu. Tato hodnota λ je v našem příkladě $\lambda = \alpha$. Potom:

$$\begin{aligned} E(X) &= \alpha X + \alpha^6 \\ L(X) &= X^2 + X + \alpha^3 \end{aligned}$$

Tím jsme dostali náš hledaný lokalizační mnohočlen $L(X)$ a pomocí Chienova vyhledávání můžeme nalézt pozice chyb:

$$\begin{aligned} L(\alpha^0) &= \alpha^0 + \alpha^0 + \alpha^3 = \alpha^3 \\ L(\alpha) &= \alpha^2 + \alpha + \alpha^3 = \alpha^5 \\ L(\alpha^2) &= \alpha^4 + \alpha^2 + \alpha^3 = \alpha^6 \\ L(\alpha^3) &= \alpha^6 + \alpha^3 + \alpha^3 = \alpha^6 \\ L(\alpha^4) &= \alpha + \alpha^4 + \alpha^3 = 0 \leftarrow \\ L(\alpha^5) &= \alpha^3 + \alpha^5 + \alpha^3 = \alpha^5 \\ L(\alpha^6) &= \alpha^5 + \alpha^6 + \alpha^3 = 0 \leftarrow \end{aligned}$$

Nalezené pozice je třeba invertovat:

$$\begin{aligned} (\alpha^4)^{-1} &= \frac{\alpha^0}{\alpha^4} = \alpha^3 = P_1 \\ (\alpha^6)^{-1} &= \frac{\alpha^0}{\alpha^6} = \alpha^1 = P_2 \end{aligned}$$

Nyní již můžeme vypočítat hodnotu hledaných chyb. Pro to potřebujeme derivaci lokalizačního mnohočlenu, což je $L'(X) = 1$:

$$\begin{aligned} M_1 &= \frac{E(P_1^{-1})}{L'(P_1^{-1}))} = \frac{E(\alpha^4)}{L'(\alpha^4)} = \frac{\alpha^3}{1} = \alpha^3 \\ M_2 &= \frac{E(P_2^{-1})}{L'(P_2^{-1}))} = \frac{E(\alpha^6)}{L'(\alpha^6)} = \frac{\alpha^4}{1} = \alpha^4 \end{aligned}$$

Tím dostáváme potřebné hodnoty pro sestavení chybového polynomu:

$$\begin{aligned}(P_1, M_1) &= (\alpha^3, \alpha^3) \\ (P_2, M_2) &= (\alpha^1, \alpha^4)\end{aligned}$$

Chybový polynom má pak tvar:

$$e(X) = \alpha^4 X + \alpha^3 X^3$$

Přičtením chybovému vektoru $e(X)$ k přijatému polynomu $r(X)$ získáme nulový vektor, což je hledaná vstupní zpráva kodéru.

3.4 Přímá metoda pro stanovení lokalizačního mnohočlenu

Pro kódy schopné opravit relativně málo chyb (maximálně $t = 2$) lze využít přímou metodu dekódování. Pro tyto kódy může být podle [5] tato metoda jednodušší alternativou než výše uvedené algoritmy. Metoda bude popsána na příkladu konkrétního kódu RS (32, 28).

V tomto případě bude systém syndromových rovnic [5]:

$$\begin{aligned} s_1 &= e(\alpha) = e_{j_1}\beta_1 + e_{j2}\beta_2 \\ s_2 &= e(\alpha^2) = e_{j_1}\beta_1^2 + e_{j2}\beta_2^2 \\ s_3 &= e(\alpha^3) = e_{j_1}\beta_1^3 + e_{j2}\beta_2^3 \\ s_4 &= e(\alpha^4) = e_{j_1}\beta_1^4 + e_{j2}\beta_2^4 \end{aligned} \tag{3.18}$$

Kde

$$\begin{aligned} \beta_1 &= \alpha^{j_1} \\ \beta_2 &= \alpha^{j_2} \end{aligned} \tag{3.19}$$

Z rovnice 3.1 podle [5] můžeme utvořit výrazy:

$$\begin{array}{lcl} s_1 s_3 & + & s_2^2 \\ s_1 s_4 & + & s_2 s_3 \\ s_2 s_4 & + & s_3^2 \end{array} \tag{3.20}$$

a násobením těchto výrazů s β_1^2, β_1 a 1, v tomto pořadí, získáme rovnici:

$$(s_1 s_3 + s_2^2)\beta_1^2 + (s_1 s_4 + s_2 s_3)\beta_1 + s_2 s_4 + s_3^2 = 0$$

Stejným způsobem, ale nyní vynásobením β_2^2, β_2 a 1 dostaneme rovnici:

$$(s_1 s_3 + s_2^2)\beta_2^2 + (s_1 s_4 + s_2 s_3)\beta_2 + s_2 s_4 + s_3^2 = 0$$

Poslední dvě rovnice jsou téměř stejné, s rozdílem, že první je vyjádřena s β_1 a druhá s β_2. Mohou však být podle [5] sloučeny do jedné rovnice jako:

$$(s_1 s_3 + s_2^2)\beta^2 + (s_1 s_4 + s_2 s_3)\beta + s_2 s_4 + s_3^2 = 0 \tag{3.21}$$

kde $\beta = \beta_1, \beta_2$ jsou dva kořeny rovnice 3.4. Kořen této rovnice lze nalézt pomocí Chienova vyhledávání přes všechny možné hodnoty β, které jsou číslovány pro tento kód RS (32, 28) od 0 do 31 ($\alpha^0 \ldots \alpha^{31}$) a vedou nás až k řešení kořenů β_1, β_2. Další možností je, že jakmile je jeden kořen určen, další dostaneme pomocí rovnice:

$$\beta_2 = \beta_1 + \frac{s_1 s_4 + s_2 s_3}{s_1 s_3 + s_2^2}. \tag{3.22}$$

Jakmile jsou hodnoty kořenů určeny, rovnice 3.1 může být vyřešena pro výpočet chybové hodnoty. V prvním případě [5]:

$$s_2 + s_1\beta_2 = e_{j_1}(\beta_1^2 + \beta_1\beta_2)$$

a tedy:

$$e_{j_1} = \frac{(s_2 + s_1\beta_2)}{(\beta_1^2 + \beta_1\beta_2)} \tag{3.23}$$

Ve druhém případě pak:

$$s_2 + s_1\beta_1 = e_{j_2}(\beta_2^2 + \beta_1\beta_2)$$

a tedy:

$$e_{j_2} = \frac{(s_2 + s_1\beta_1)}{(\beta_2^2 + \beta_1\beta_2)} \tag{3.24}$$

Pokud je počet chyb v přijatém vektoru roven dvěma, pak je řešení jedinečné. Oprava chyb se pak provádí přidáním přijatého vektoru k chybovému vektoru.

Složitost této přímé metody je menší než u Euklidova algoritmu, protože rovnice 3.4 je přímo získána s komponenty syndromového vektoru a její kořeny jsou lokátory chyb β_1 a β_2, jejichž výpočet vede k celému řešení systému. Není zde třeba hledat žádný chybový polynom. Použití této metody je ovšem omezeno na kódy s korekční schopností $t \leq 2$.

3.4.1 Příklad použití přímé metody na dekódování

Pro lepší pochopení metody opět uvedeme příklad. Máme přijatou zprávu r = (000 111 000 101 000 000 000). Posloupnost byla zakódována kódem RS (7, 3) definován polem GF (2^3)nad základním polynomem $p_i(X) = 1 + X^2 + X^3$.

Nejprve si přepíšeme přijatou zprávu do polynomiálního tvaru:

$$r(X) = \alpha^4 X + \alpha^3 X^3$$

Dalším krokem je výpočet syndromů.

$$\begin{aligned}
s_1(\alpha) &= r(\alpha) = \alpha^5 + \alpha^6 = \alpha^3 \\
s_2(\alpha) &= r(\alpha^2) = \alpha^6 + \alpha^2 = \alpha \\
s_3(\alpha) &= r(\alpha^3) = 1 + \alpha^5 = \alpha \\
s_4(\alpha) &= r(\alpha^4) = \alpha + \alpha = 0
\end{aligned}$$

Nyní můžeme dosadit do vztahu 3.21:

$$
\begin{aligned}
&(s_1 s_3 + s_2^2)\beta^2 + (s_1 s_4 + s_2 s_3)\beta + s_2 s_4 + s_3^2 \\
&= (\alpha^3\alpha + \alpha^2)\beta^2 + (\alpha^3 \cdot 0 + \alpha\alpha)\beta + \alpha \cdot 0 + \alpha^2 \\
&= \alpha^5\beta^2 + \alpha^2\beta + \alpha^2 \\
&= 0
\end{aligned}
$$

$\beta = \beta_1, \beta_2$ jsou dva kořeny tohoto polynomu a můžeme je najít pomocí Chienova vyhledávání. To nám ukazují následující výpočty:

$$
\begin{aligned}
L(\alpha^0) &= \alpha^5(\alpha^0)^2 + \alpha^2\alpha^0 + \alpha^2 = \alpha^5 \\
L(\alpha) &= \alpha^5\alpha^2 + \alpha^2\alpha + \alpha^2 = 0 \rightarrow \beta_1 \\
L(\alpha^2) &= \alpha^5(\alpha^2)^2 + \alpha^2\alpha^2 + \alpha^2 = \alpha^4 \\
L(\alpha^3) &= \alpha^5(\alpha^3)^2 + \alpha^2\alpha^3 + \alpha^2 = 0 \rightarrow \beta_2 \\
L(\alpha^4) &= \alpha^5(\alpha^4)^2 + \alpha^2\alpha^4 + \alpha^2 = \alpha^2 \\
L(\alpha^5) &= \alpha^5(\alpha^5)^2 + \alpha^2\alpha^5 + \alpha^2 = \alpha^4 \\
L(\alpha^6) &= \alpha^5(\alpha^6)^2 + \alpha^2\alpha^6 + \alpha^2 = \alpha^5
\end{aligned}
$$

Nalezli jsme tedy dva kořeny $\beta_1 = \alpha$ a $\beta_2 = \alpha^3$, to znamená, že první chyba je na pozici $j_1 = 1$ a druhá chyba je na pozici $j_2 = 3$. Nyní již můžeme pomocí vztahů 3.23 a 3.24 určit hodnotu chyb:

$$
\begin{aligned}
e_{j_1} &= \frac{(s_2 + s_1\beta_2)}{(\beta_1^2 + \beta_1\beta_2)} = \frac{\alpha + \alpha^6}{\alpha^2 + \alpha^4} = \frac{\alpha^2}{\alpha^5} = \alpha^4 \\
e_{j_2} &= \frac{(s_2 + s_1\beta_1)}{(\beta_2^2 + \beta_1\beta_2)} = \frac{\alpha + \alpha^4}{\alpha^6 + \alpha^4} = \frac{\alpha^3}{1} = \alpha^3
\end{aligned}
$$

Chybový polynom tedy bude ve tvaru:

$$e(X) = \alpha^4 X + \alpha^3 X^3$$

Po opravě je tedy zřejmé, že původní zpráva byl nulový vektor.

4 SROVNÁNÍ ALGORITMŮ DEKÓDOVÁNÍ

Výše uvedené metody dekódování, Berlekamp-Masseyův algoritmus, Euklidův algoritmus, Peterson-Gorenstein-Zierler algoritmus a přímá metoda, mají odlišné vlastnosti. Můžeme je porovnávat na základě výpočetní náročnosti, korekční schopnosti a dosažené chybovosti FER, tato srovnání budou uvedena v této kapitole.

4.1 Výpočetní náročnost jednotlivých metod

Výpočetní náročnost jednotlivých dekódovacích metod je závislá především na korekční schopnosti t dekódovaného kódu. Následující tabulka porovnává přibližný počet základních výpočetních operací pro stanovení lokalizačního mnohočlenu danou metodou. Uvedené počty operací jsou pouze pro stanovení lokalizačního mnohočlenu, operace ostatních částí dekódování nejsou zahrnuty, protože je lze použít u všech metod stejné a tedy se v nich algoritmy neliší. Přibližné srovnání je uvedeno v tabulce 4.1.

Dekódovací metoda	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$
Přímá metoda	5	9	–	–	–
BM algoritmus	6	24	36	96	150
Euklidův algoritmus	6	25	36	98	150
PGZ	2	8	27	64	125

Tab. 4.1: Přibližný počet základních operací v závislosti na korekční schopnosti.

Jak lze vidět z tabulky, přímá metoda je méně náročná, nicméně lze ji použít pouze pro kódy s korekční schopností $t \leq 2$. Její náročnost v závislosti na t lze přibližně vyjádřit vztahem $3t$. BM a Euklidův algoritmus jsou na tom s výpočetní náročností přibližně stejně, to lze vidět i v tabulce 4.1. Obecně lze jejich výpočetní náročnost v závislosti na t vyjádřit jako $6t^2$ [12]. U PGZ algoritmu můžeme počet operací vyjádřit přibližně jako t^3, z čehož vyplývá, že je na tom lépe s výpočetní

náročností než BM a Euklidův algoritmus přibližně do $t = 6$. Hodnoty počtu operací uvedené v tabulce jsou přibližné, mohou se částečně lišit pro různé vstupní posloupnosti, především pak pro počet vyskytnutých chyb v dané posloupnosti.

4.2 Korekční schopnost

Jak už bylo zmíněno výše, přímou metodu lze využít pouze pro dekódování RS kódu schopného opravit $t = 2$ chyby. Z tohoto hlediska je jasně nejhorším, avšak pro kódy s $t \leq 2$ je vhodný pro svou jednoduchost a malou výpočetní náročnost.

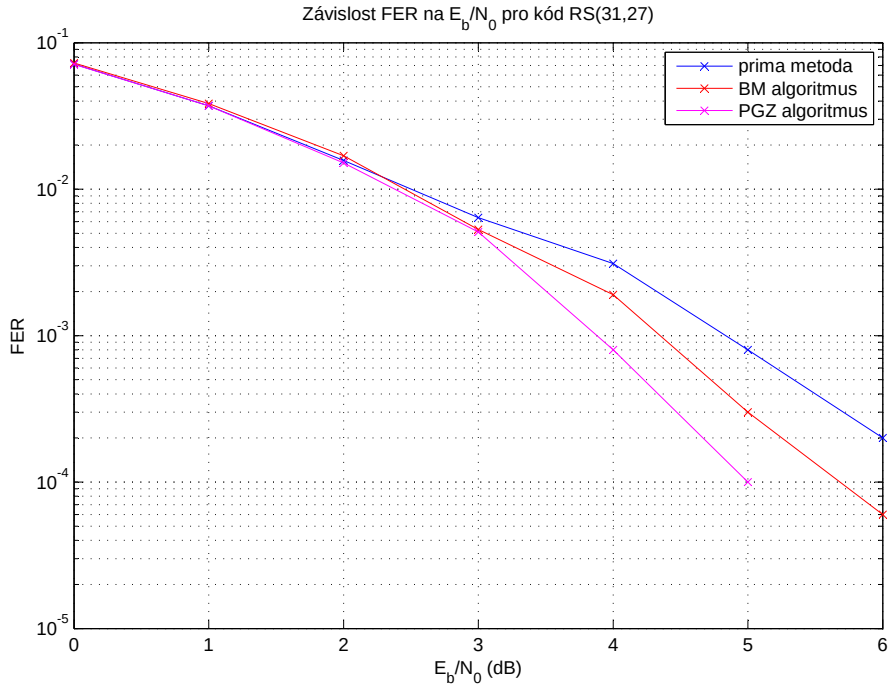
PGZ algoritmus bývá používán pro $t \leq 6$, pro vyšší t již není jeho použití vhodné a převážně se využívá BM nebo Euklidova algoritmu.

4.3 Frame Error Rate (FER)

Jelikož RS kódy jsou nebinárními kódy, nevztahuje se k nim BER (Bit Error Rate), ale zjišťuje se závislost FER (Frame Error Rate) na E_b/N_0, tedy na poměru energie, kterou jsme "průměrně" vynaložili na vyslání jednoho bitu dat ku spektrální výkonové hustotě šumu. Všechny závislosti vynesené do grafu v této kapitole jsou výstupy ze simulačního programu, který byl vytvořen v rámci této práce. Grafy se pouze přibližují teoretickým hodnotám, odchylky jsou způsobeny především simulováním na ne úplně dostatečném počtu vstupních symbolů a to z důvodu velmi zdlouhavého počítání Matlabu, kdy například simulace pro $3 \cdot 10^4$ vstupních symbolů trvala přibližně 24 hodin pro jeden kód.

Na obrázku 4.1 je graf znázorňující závislost FER na E_b/N_0 pro kód RS(31,27). Simulace proběhla pro přenos $2 \cdot 10^4$ symbolů a pro rozsah E_b/N_0 0 – 8 dB. Grafy vyšly přibližně podle předpokladů, protože dekódování kódu RS(31,27) s korekční schopností $t = 2$ by měly zajistit všechny metody s přibližně stejnou chybovostí.

Jako další graf jsem zvolil závislost FER na E_b/N_0 kódů RS(255,239), RS(255,223) a RS(248,216) dekódované pomocí BM algoritmu. Jedná se o kódy, které se používají v zabezpečení FEC (Forward Error Correction) u specifikací optických pasivních sítí. RS(255,239), který má parametry $t = 8$, $R = 0,937$ je povinně použit u standartu

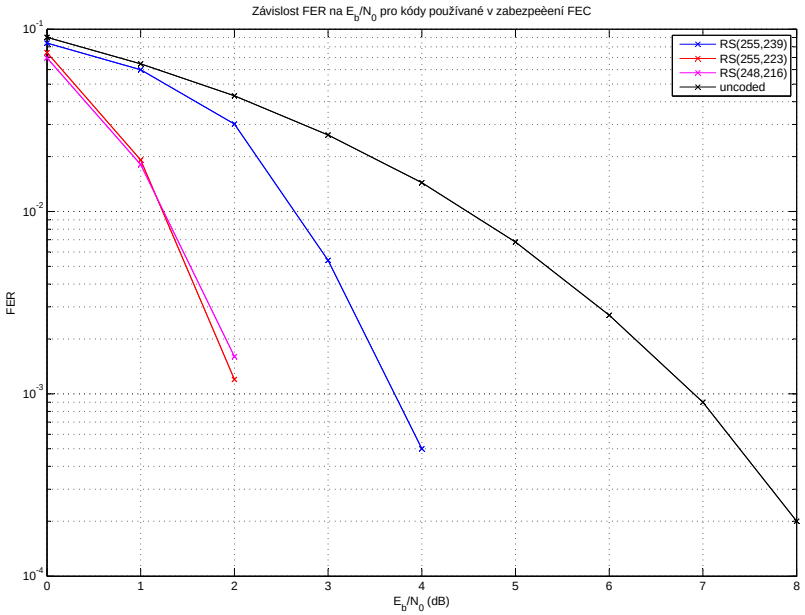


Obr. 4.1: Závislost FER na Eb/N0 pro kód RS(31,27) dekódovaný vybranými metodami.

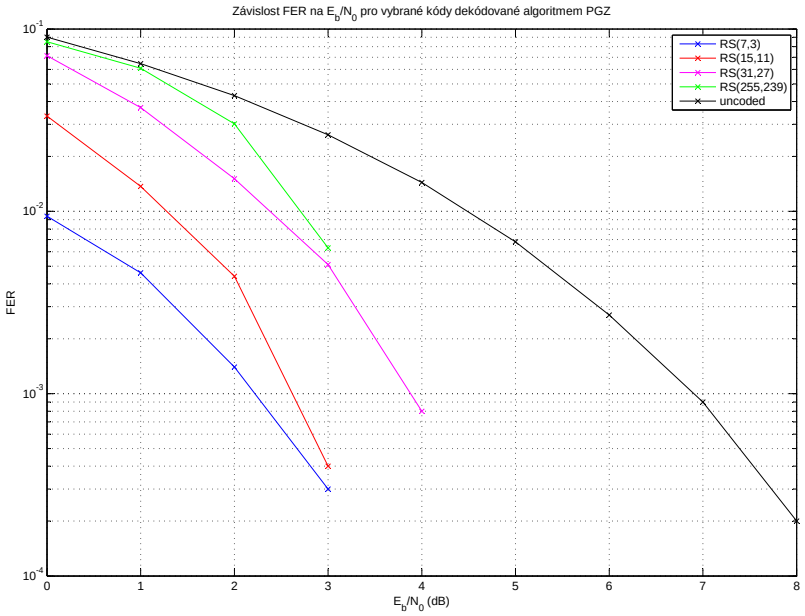
GPON a volitelně pak u EPON, RS(255,223), $t = 16$, $R = 0,875$ je ve standartu 10GEPON a RS(248,216), $t = 16$, $R = 0,871$ v XG-PON. Jak lze vidět z grafů na obrázku 4.2, všechny tři kódy zajistily nulovou chybovost od hodnoty $E_b/N_0 > 4dB$, RS(255,223) a RS(248,216) pak zajistili přenos bez chyb již pro $E_b/N_0 > 2dB$. Pro simulaci těchto kódů byl použit pro dekódování BM algoritmus, protože přímá metoda již není tyto kódy vůbec schopná dekódovat, a u PGZ algoritmu se pro tak vysokou korekční schopnost velmi navýšil čas výpočtu, v Matlabu to byl až desetinásobek času výpočtu pro BM algoritmus.

Další graf viz obrázek 4.3 ukazuje závislost FER na E_b/N_0 pro různé délky kódů dekódované algoritmem PGZ. Jak lze vidět, kratší kódy vykazují menší chybovost, ač mají menší korekční schopnost. To je pochopitelně způsobeno kratší kódovou délkou a tedy menší pravděpodobností výskytu většího počtu chyb v kódové délce.

Simulace dalších kódů je možno nalézt ve výukovém programu, případně si v něm libovolnou simulaci nasimulovat podle zvolených parametrů.



Obr. 4.2: Závislost FER na Eb/N0 pro kódy používané v zabezpečení FEC.



Obr. 4.3: Závislost FER na Eb/N0 pro různé kódy dekódované pomocí PGZ.

5 IMPLEMENTACE V MATLABU

Tato kapitola obsahuje popis výukového programu vytvořeného na základě prostudování problematiky Reed Solomonových kódů a metod jejich dekódování. Program je vytvořen v programovém prostředí Matlab. Cílem bylo naprogramovat vybrané algoritmy dekódování a umožnit s pomocí programu demonstrovat jejich vzájemné odlišnosti. Pro realizaci jsem vybíral algoritmy algebraického dekódování, všechny zvolené metody jsou teoreticky rozebrány ve 3. kapitole. Co se týká pravděpodobnostího dekódování RS kódů, toto řešení není moc rozšířené a není k němu dostupná potřebná literatura, na které by se dalo stavět. V programu jsou tedy realizovány algoritmy: Berlekamp-Masseyův, Peterson-Gorenstein-Zierleův a přímá metoda. Měl jsem v plánu realizovat i Euklidův algoritmus, bohužel se mi ho nepodařilo pořádně odladit a nepracoval úplně správně, proto jsem ho nakonec do finálního programu neimplementoval.

Největším úskalím implementace jednotlivých metod dekódování v Matlabu byla reprezentace Galoisova pole. Prozkoušel jsem několik různých funkcí pro počítání s prvkami GF, snažil jsem se i o tvorbu vlastního, nakonec jsem však zůstal u využití přímo funkce implementované Matlabem. Tato funkce až na některé chyby při různých operacích s prvky GF pracovala bez problémů. Jediným problémem je reprezentace výsledku, kdy výstup je v podstatě číslo řádku na kterém se daný prvek nachází v tabulce GF. Například podle tabulky B.1 prvek α^3 bude na výstupu z Matlabu jako: " $ans = GF(2^4)$ array. Primitive polynomial = $D^4 + D + 1$ (19 decimal) Array elements = 4" což znamená, že jde o prvek na 4. řádku tabulky GF(16) nad polynomem $p_i(X) = 1 + X + X^4$.

Na obrázku 5.1 vidíme okno programu po spuštění, to je členěno do několika částí, které podrobněji popíšu.

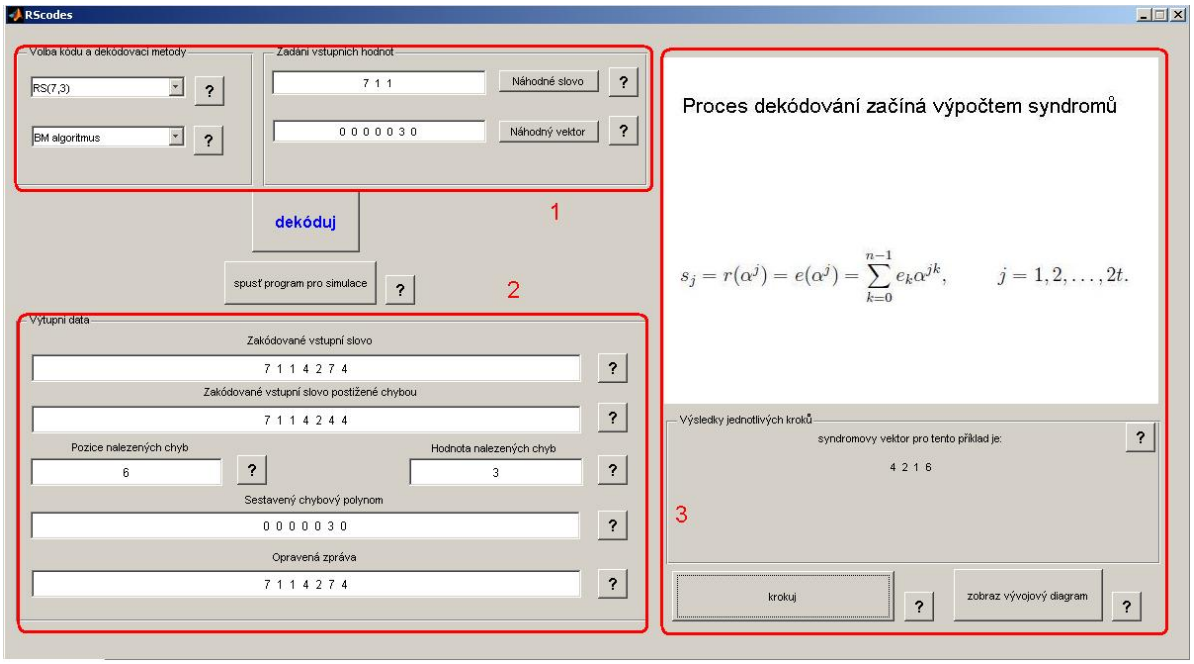
5.1 Blok zadání vstupních hodnot

První část je oddíl pro zadání vstupních hodnot. Volí se zde použitý kód a metoda jeho dekódování. Jsou zde přednastaveny 4 druhy kódů, zvolil jsem kódy s krátkými vstupními bloky vstupních slov, z důvodu přehlednosti výstupních dat. Je zde

možno vybrat z kódů RS(7,3), RS(15,11), RS(15,9) a RS(31,27). Dále je nutno zadat posloupnost vstupního slova a chybový vektor, ty může uživatel zadat sám nebo vygenerovat náhodné posloupnost pomocí tlačítka "Náhodné slovo" respektive "Náhodný vektor". Po vyplnění příslušných vstupních dat se dekódování provede stiskem tlačítka "dekóduj".

5.2 Výstupní data dekódování

Druhá část obsahuje výstupní data celého procesu dekódování. Pro názornost je zde zobrazeno Výstupní kódové slovo, což je posloupnost vystupující z kodéru. Další položka vypisuje tuto posloupnost zasaženou chybami. Následující okna slouží pro výpis nalezených pozic a hodnot chyb, z nichž je sestrojen chybový polynom $e(X)$ zobrazený v okně pod nimi. Poslední položkou v této části je dekódovaná zpráva. V případě, že zvolené dekódování proběhlo v pořádku a byly opraveny všechny chyby, je zpřístupněna třetí část.



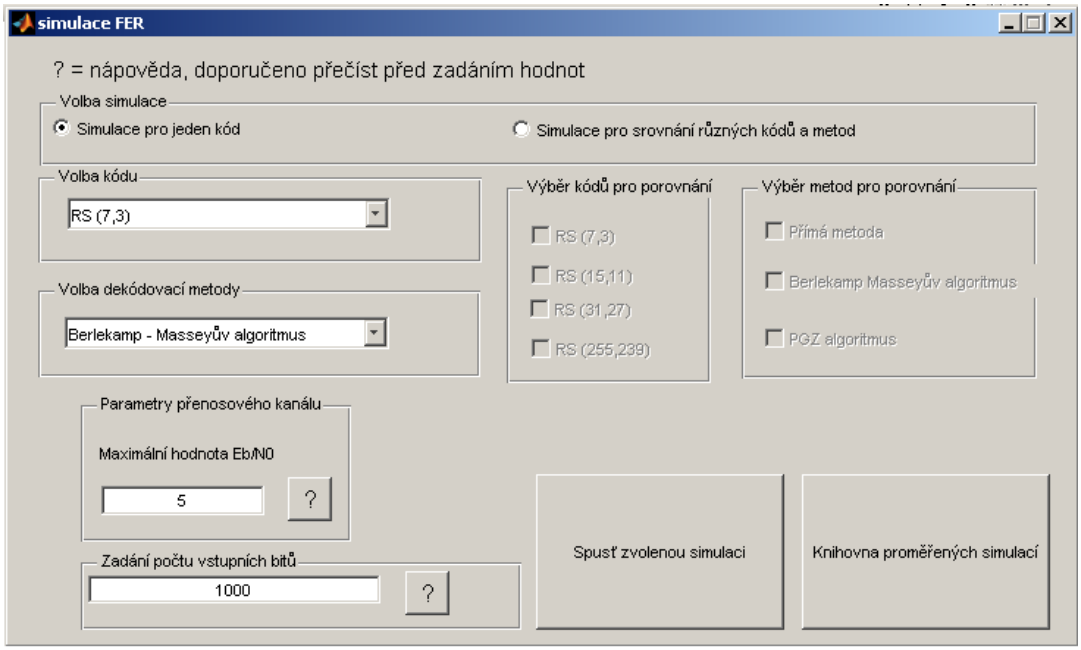
Obr. 5.1: Ukázka výukového programu v prostředí Matlab.

5.3 Krokování

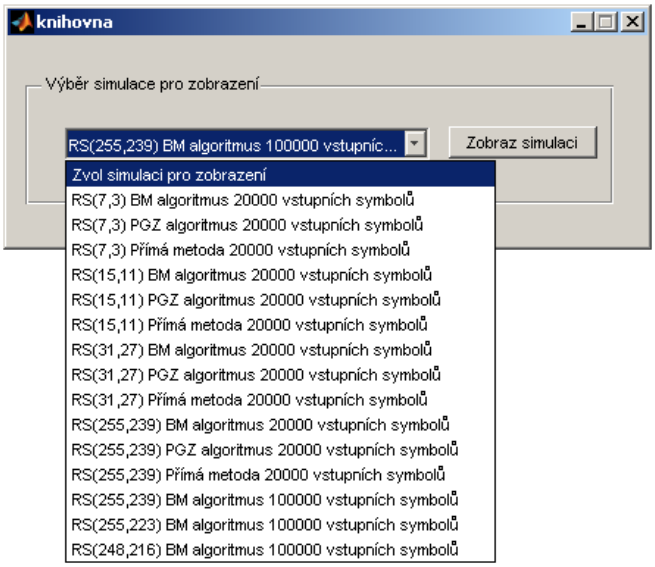
Tato část umožňuje projít po krocích dekódování, které bylo provedeno uživatelem zvolenou metodou. Zde jsem volil, dle mé úvahy, nejdůležitější kroky daného algoritmu. Na základě krokování by měl uživatel získat přehled každé z imlementovaných metod. Ke klíčovým rovnicím v krokování jsou vždy uvedeny i výsledky daných rovnic pro zvolený příklad. Jediným problémem je reprezentace výsledku Galoisova pole v Matlabu, kdy výsledky z jednotlivých kroků jsou většinou právě hodnoty prvků z GF.

5.4 Simulační část

Z hlavního okna výukového programu pomocí tlačítka "Program pro simulace" lze spustit program, který umožňuje simulovat závislost FER na E_b/N_0 pro zvolený kód a metodu viz obrázek 5.2. Lze zde simulovat buď jednotlivé kódy do jednoho grafu, nebo lze zvolit i simulace kde je vykreslená závislost FER na E_b/N_0 pro několik kódů nebo různých metod dekódování najednou, vhodných pro porovnání. Toto je možné zvolit v bloku "výběr simulace". Pro simulaci jsou zde na výběr přednastaveny kódy: RS(7,3), RS(15,11), RS(31,27) a také kód používaný v zabezpečení FEC v optických standartech RS(255,239). Jako přenosový kanál je zde použit AWGN (Additive white Gaussian noise) kanál, vytvořený pomocí implementovaných funkcí programu Matlab. Aby byla zvolená simulace co nejpřesnější, je vhodné zvolit velký počet vstupních symbolů, ideálně to je aspoň $2 \cdot 10^4$ vstupních symbolů. To však vyžaduje na běžném počítači spoustu hodin pro provedení simulace, proto jsem se rozhodl do této části přidat i knihovnu již provedených simulací, kde jsou pro ukázku některé mnou změřené simulace, viz obrázek 5.3.



Obr. 5.2: Ukázka simulačního programu v prostředí Matlab.



Obr. 5.3: Ukázka knihovny simulačního programu.

6 ZÁVĚR

Cílem této diplomové práce bylo nastudovat problematiku protichybového zabezpečení pomocí Reed-Solomonova kódu a především se zaměřit na jednotlivé přístupy dekódování a následně pak dané metody porovnat. Následně pak po rozebrání teorie v programovém prostředí Matlab vytvořit program, kde budou jednotlivé metody dekódování implementovány a bude možnost jejich porovnání.

První část zadání, teoretický rozbor celé problematiky, je zpracována v kapitolách 1 - 3. První kapitola je věnována úvodu do problematiky protichybového kódování, jsou zde uvedeny základní pojmy z této oblasti a jsou zde popsány základy z algebry jako je konečné a Galoisovo těleso, které jsou nezbytné pro problematiku RS kódů. V této kapitole jsou také popsány základní vlastnosti RS kódu a jeho samotné kódování.

Druhá kapitola se zabývá obecným pohledem na dekódování RS kódu, kde jsou popsány jednotlivé kroky obecného přístupu algebraického dekódování RS kódů. Tyto zobecněné kroky bývají pak různě řešeny na základě jednotlivých algortimů pro dekódování, vybrané přístupy jsou popsány v kapitole třetí. Je zde popsán Berlekamp-Masseyův, Euklidův a Peterson-Gorenstein-Zierleův algoritmus a dále pak přímá metoda pro dekódování RS kódů. U každé metody je pro lepší pochopení podrobně uveden konkrétní příklad dekódování.

Vlastním výsledkům práce je věnována 4. a 5. kapitola. Ve 4. kapitole jsou porovnány jednotlivé metody dekódování a to na základě výpočetní náročnosti, korekční schopnosti a pak dosažené výsledky FER. Tato porovnání vycházejí částečně z teoretických údajů a z větší části ze simulačního programu vytvořeného v prostředí Matlab. V 5. kapitole je prezentován nástroj vytvořený v programovém prostředí Matlab, prezentující uvedené principy a umožňující porovnání uvedených algoritmů. Vytvořený program je koncipován tak, aby ho bylo možné použít jako výukovou pomůcku, jak je dáno zadáním. Pomocí krokování jsou jednotlivé algoritmy názorně demonstrovány. Veškeré položky v grafickém rozhraní jsou opatřeny nápovědou tak, aby uživatel věděl, jak má postupovat. Věřím, že tímto byly splněny všechny cíle zadání.

LITERATURA

[1] BERLEKAMP, E. Algebraic Coding Theory. New York McGraw-Hill, 1968

[2] ČÍKA, P. Kodek BCH kódu. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2005.

[3] ČÍKA, P., KOTON, J., KŘIVÁNEK, V. Samoopravné Reed-Solomonovy kódy 2006, [cit. 2011-03-16] Dostupné z URL: http://access.feld.cvut.cz/view.php?cisloclanku=2006080002.

[4] DUŠEK, J. Návrh a implementace opravných kódů pro systém Orpheus Praha: ČVUT FEL katedra počítačů 2007.

[5] FARELL, P.G., MOREIRA, J.C. Essentials of Error-Control Coding. John Wiley, 2006, ISBN-13 978-0-470-02920-6.

[6] GATHEN, J., GERHARD, J. Modern Computer Algebra. Cambridge: Cambridge University Press, 1999.

[7] GLAVIEUX, A. Channel Coding in Communication Networks: From Theory to Turbocodes. Wiley-ISTE, 2007, ISBN: 978-1-90520-924-8.

[8] HANZO, L., LIEW, T.H., YEAP, B.L. Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmision over Fading Channels. JohnWiley, 2002, ISBN: 0470847263.

[9] HOFFMAN, D.G., LEONARD, D.A, LINDNER, C.C, PHELPS, K.T., RODGER, C.A., WALL, J.R. Coding Theory The Essentials. New York, Basel, Hong Kong: Marcel Dekker Inc., 1991. ISBN 0-8247-8611-4

[10] HUFFMAN, W.C., LEONARD, D.A, LINDNER, C.C, PHELPS, K.T., RODGER, C.A., WALL, J.R. Fundamentals of Error-Correcting Codes. United Kingdom: Cambridge University Press, 2003. ISBN 0-521-78280-5.

[11] LIN, S., COSTELLO, D.J. Error Control Coding: Fundamentals and Applications, second edition. Prentice Hall: Englewood Cliffs, NJ, 2005, ISBN: 0-13-042672-5.

[12] MOON, T.K. Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, 2005, ISBN-13: 978-0070010697.

[13] MORELOS, R.H. The Art of Error Correcting Coding Wiley-Interscience, 2002, ISBN: 0471 49581 6.

[14] NĚMEC, K. Datová komunikace. Skripta. VUT FEKT, Brno 2007.

[15] PURSER, M. Introduction to Error-correcting codes. Boston, London: Artec House, 1995. ISBN 0-89006-784-8.

[16] SKALAR, B. Digital Communications, Fundamentals and applications Prentice-Hall, 2003, ISBN 0-13-084788-7.

[17] ZAPLATÍLEK, K., DOŇAR, B. MATLAB - tvorba uživatelských aplikací BEN - technická literatura, Praha 2004, ISBN 80-7300-133-0

[18] ZAPLATÍLEK, K., DOŇAR, B. MATLAB - začínáme se signály BEN - technická literatura, Praha 2006, ISBN 80-7300-200-0

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BER Bit Error Rate – bitová chybovost

FER Frame Error Rate – rámcová chybovost

GF Galoisovo pole – Galois Field

LSFR Linear Shift Feedback Register – lineární posuvný zpětnovazební registr

d odhad chyby

d_{min} minimální kódová vzdálenost

k počet vstupních symbolů do kodéru

M_l hodnota chyby

n počet výstupních symbolů z kodéru

ν očekávaný počet chyb

s_j syndromy přijatých dat

t korekční schopnost kódu

X_l lokátor chyby

Z_p konečné těleso

$A(X)$ pomocný chybový mnohočlen

$c(X)$ výstupní kódové slovo

$e(X)$ chybový polynom

$E(X)$ polynom pro určení hodnoty chyb

$g(X)$ vytvářecí mnohočlen

$i(X)$ vstupní kódové slovo

$p(X)$ paritní polynom

$r(X)$ polynom přijatý dekodérem

$S(X)$ syndromový vektor

$T(X)$ dočasný chybový mnohočlen

$L(X)$ lokalizační mnohočlen

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

Přiložené CD obsahuje jednotlivé soubory M-file pro vytvořený výukový program v prostředí Matlab a také program zkompilovaný do exe souboru.

V kořenovém adresáři CD se nacházejí dvě složky, složka RScodes_aplikace, složka Zdrojové kódy a pak soubor s elektronickou verzí diplomové práce. V první složce je umístěná spustitelná aplikace RScodes.exe, přesná cesta je RScodes_aplikace/src/RScodes.exe. Druhá složka obsahuje jednotlivé M-file soubory se zdrojovými kódy a veškeré další soubory použité ve výukovém programu.

Ke spuštění programu je potřeba mít v PC nainstalovaný program Matlab.

B POUŽITÉ GALOISOVA POLE PRO KÓDOVÁNÍ A DEKÓDOVÁNÍ VYBRANÝCH KÓDŮ

Exponenciální tvar	Polynomiální tvar	Binární tvar
0	0	0 0 0 0
α^0	1	1 0 0 0
α	α	0 1 0 0
α^2	α^2	0 0 1 0
α^3	α^3	0 0 0 1
α^4	$\alpha + 1$	1 1 0 0
α^5	$\alpha^2 + \alpha$	0 1 1 0
α^6	$\alpha^3 + \alpha^2$	0 0 1 1
α^7	$\alpha^3 + \alpha + 1$	1 1 0 1
α^8	$\alpha^2 + 1$	1 0 1 0
α^9	$\alpha^3 + \alpha$	0 1 0 1
α^{10}	$\alpha^2 + \alpha + 1$	1 1 1 0
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	0 1 1 1
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$	1 1 1 1
α^{13}	$\alpha^3 + \alpha^2 + 1$	1 0 1 1
α^{14}	$\alpha^3 + 1$	1 0 0 1

Tab. B.1: Galoisovo pole GF(16) generované polynomem $p_i(X) = 1 + X + X^4$

Exponenciální tvar	Polynomiální tvar	Binární tvar
0	0	0 0 0 0 0
α^0	1	1 0 0 0 0
α	α	0 1 0 0 0
α^2	α^2	0 0 1 0 0
α^3	α^3	0 0 0 1 0
α^4	α^4	0 0 0 0 1
α^5	α^2+1	1 0 1 0 0
α^6	$\alpha^3+\alpha$	0 1 0 1 0
α^7	$\alpha^4+\alpha^2$	0 0 1 0 1
α^8	$\alpha^3+\alpha^2+1$	1 0 1 1 0
α^9	$\alpha^4+\alpha^3+\alpha$	0 1 0 1 1
α^{10}	α^4+1	1 0 0 0 1
α^{11}	$\alpha^2+\alpha+1$	1 1 1 0 0
α^{12}	$\alpha^3+\alpha^2+\alpha$	0 1 1 1 0
α^{13}	$\alpha^4+\alpha^3+\alpha^2$	0 0 1 1 1
α^{14}	$\alpha^4+\alpha^3+\alpha^2+1$	1 0 1 1 1
α^{15}	$\alpha^4+\alpha^3+\alpha^2+\alpha+1$	1 1 1 1 1
α^{16}	$\alpha^4+\alpha^3+\alpha+1$	1 1 0 1 1
α^{17}	$\alpha^4+\alpha^3+\alpha^2+1$	1 1 0 0 1
α^{18}	$\alpha+1$	1 1 0 0 0
α^{19}	$\alpha^2+\alpha$	0 1 1 0 0
α^{20}	$\alpha^3+\alpha^2$	0 0 1 1 0
α^{21}	$\alpha^4+\alpha^3$	0 0 0 1 1
α^{22}	$\alpha^4+\alpha^2+1$	1 0 1 0 1
α^{23}	$\alpha^3+\alpha^2+\alpha+1$	1 1 1 1 0
α^{24}	$\alpha^4+\alpha^3+\alpha^2+\alpha$	0 1 1 1 1
α^{25}	$\alpha^4+\alpha^3+1$	1 0 0 1 1
α^{26}	$\alpha^4+\alpha^2+\alpha+1$	1 1 1 0 1
α^{27}	$\alpha^3+\alpha+1$	1 1 0 1 0
α^{28}	$\alpha^4+\alpha^2+\alpha$	0 1 1 0 1
α^{29}	α^3+1	1 0 0 1 0
α^{30}	$\alpha^4+\alpha$	0 1 0 0 1

Tab. B.2: Galoisovo pole GF(32) generované polynomem $p_i(X) = 1 + X^2 + X^5$