UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Kodér/dekodér BCH kódů

Lukáš Hanc

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš Hanc**
Osobní číslo:	**I10278**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Komunikační a mikroprocesorová technika**
Název tématu:	**Kodér/dekodér BCH kódů**
Zadávající katedra:	**Katedra elektrotechniky**

Zásady pro vypracování:

V teoretické části proveďte analýzu BCH kódů z hlediska jejich vlastností a využití v praxi. Součástí teoretické části bude popis vlastního kódování a dekódování (kodéru/dekodéru).
V praktické části vytvořte SW modul umožňující pro zadanou či importovanou zprávu kódování/ dekódování této zprávy se zobrazením jednotlivých kroků kódování/dekódování. Součástí SW bude i možnost zanést chybu vzniklou při přenosu, čímž lze ověřit schopnosti detekce/opravy chyb daného kódu.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Adámek, J.: Kódování, SNTL, Praha, 1989
Couch, L. W.: Digital and Analog Communication systems, 7 th edition, Prentice Hall, 2007
Vlček, K.: Komprese a kódová zabezpečení v multimediálních komunikacích, Praha, BEN, technická literatura, 2004, ISBN 80-86056-68-6
Dobeš, J., Žalud, V.: Moderní radiotechnika, BEN, Praha 2006
S., Lin, Costello, D.J.: Error Control Coding, 2ed, Perason edu, 2004

Vedoucí bakalářské práce: **Ing. Jan Pidanič, Ph.D.**
Katedra elektrotechniky

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Zdeněk Němec, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 08.05.2013 Lukáš Hanc

Poděkování

Rád bych poděkoval vedoucímu této Bakalářské práce panu Ing. Janu Pidaničovi, Ph.D. za odborné vedení, konzultace a podnětné návrhy během tvorby této práce. Také bych rád poděkoval svým rodičům za podporu po celou dobu práce.

Anotace

Tato bakalářská práce se zabývá skupinou bezpečnostních kódů, které jsou schopny zabezpečit zpracovávaná data proti nezávislým chybám. Jsou zde uvedeny podklady z oboru lineární algebry, jako například struktura zvaná Galoisova tělesa, kterou právě BCH kódy využívají. V této práci se uvádí vlastnosti těchto kódů a princip kódování a dekódování. Pro tuto práci byl vybrán Petersonův dekódovací algoritmus. Také je zde uveden příklad zakódování a dekódování zprávy. Na závěr práce je uveden popis a funkce aplikace pro simulování jednotlivých kroků kódování zprávy, vzniku chyby a dekódování zprávy. Tato aplikace je právě předmětem této bakalářské práce.

Klíčová slova

BCH kód, kódování, dekódování, MATLAB

Title

Encoder/decoder of BCH codes

Annotation

This bachelor thesis deals with a group of security codes that are able to provide processed data against independent errors. There are listed documents in the field of linear algebra, such as structure called Galois field, which BCH codes are using. The properties of these codes and the principle of encoding and decoding are described there. For this work was chosen Peterson's decoding algorithm. There is also an example of encoding and decoding messages. The last part contains the description and function of application for simulating each step message encoding, generating errors and decoding message. This application is currently the subject of this bachelor thesis.

Keywords

BCH code, encoding, decoding, MATLAB

Obsah

Seznam zkratek

ASCII	American Standard Code for Information Interchange
DVD	Digital Video Disc
BRD	Blue Ray Disc
CD	Compact Disc
BC	Boseho a Chaudhuriho
GF	Galois Field
SNR	Signal to Noise Ratio
PC	Personal computer

Seznam obrázků

Seznam tabulek

Úvod

Z důvodu zavádění digitalizace různých datových přenosů, jako jsou telefonní linky, sítě Wi-Fi a podobně, se do popředí začal dostávat obor vědy zabývající se kódováním, který slouží k zabezpečení dat při přenosu. Tento vědní obor neřeší přenos informace pouze mezi dvěma výpočetními stanicemi, ale mezi jakýmikoliv body zpracovávajícími informaci. Předmětem kódování mohou být různé požadavky, jako je zabezpečení přenosu, komprese přenášené informace, atd. Tato bakalářská práce se zabývá právě požadavkem na zabezpečení informace po stránce vzniku chyb během přenosu na přenosovém kanálu.

Z historického hlediska se teorií kódování začal zabývat C. E. Shannon (Bellovy laboratoře – 1942), kde se zabýval především teorií informace, jenž je nemálo důležitá pro konstrukci kódů. V této době byly také zkonstruovány první lineární kódy (Adámek, 1989), pány Hammingem a Golayem. Na jejich práci navázali Reed a Muller, kteří zkonstruovali princip kódů, u kterých je možné předem nadefinovat libovolný počet chyb. Následně v roce 1959 byly vyvinuty BCH kódy (Adámek, 1989), které mají jako Reed-Mullerovy libovolný počet opravitelných chyb, ovšem s výhodnějšími parametry. Tyto parametry jsou velká kódová vzdálenost pro různé počty informačních znaků. BCH je zkratka konstruktérů, což jsou A. Hocquenghem, R. C. Bose a D. K. Ray-Chaudhuri. (Adámek, 1989). Tyto BCH kódy budou v této práci rozebrány a vysvětleny principy kódování a dekódování.

V první kapitole budou nejprve popsány právě bezpečnostní kódy, do kterých BCH kódy spadají. Obecné vlastnosti bezpečnostních kódů a některé prvky, které jsou pro jejich konstrukci použity. Také zde budou nadefinovány pojmy, které se vyskytují v oboru kódování.

Další kapitola se pak bude zabývat oblastí lineární algebry, která je velmi využívaná při kódování. Zde budou vysvětleny pojmy jako polynom, Těleso a hlavně teorie Galoisových těles, které hrají významnou roli právě u BCH kódů.

Třetí kapitola se zabývá už konkrétním popisem BCH kódů. Jejich zařazením a vlastnostmi. Dále je v této kapitole popsán obecný princip kódování a dekódování. Ačkoliv existuje celá řada algoritmů využívaných pro dekódování, pro tuto práci byl vybrán Petersonův algoritmus.

V předposlední kapitole je ukázán na konkrétním příkladu postup zakódování a dekódování konkrétně nadefinované zprávy. Nejprve je ukázán postup výběru kódu dle požadavků, následně postup zakódování a zavedení chyby. Nakonec je tato chybná zpráva pomocí popsaného Petersonova algoritmu dekódována.

Poslední kapitola se zabývá popisem aplikace, která byla předmětem této práce. Jsou zde vysvětleny jednotlivé prvky aplikace a její použití.

1 Kódování

1.1 Úvod

Většina výpočetních zařízení využívají při kódování binární soustavu pro ukládání informací. Takto uložená informace má největší odolnost SNR. SNR označuje vztah úrovně signálu a úrovně šumu. Je definován jako poměr výkonu signálu k výkonu šumu. Binární kódy mají největší SNR z důvodu použití pouze dvou úrovní signálu, při vyjádření informace. Pokud by byl použit například ternární kód, sníží se kvantizační úroveň a tím by se snížil i SNR. Z důvodu majoritního zastoupení binárních kódů se tato práce zabývá konstrukcí pouze binárních BCH kódů. Kódování je proces, který převádí informaci z jedné formy do druhé. Této novou formou vyjádřené informaci se říká kódové slovo. Kódování využíváme z mnoha důvodů, jako jsou zabezpečení informace, komprimace informace a další.

Kódování bychom mohli rozdělit do tří základních dělení podle jejich účelu.

- Zdrojové kódování
- Kanálové kódování
- Šifrování

Zdrojové kódování:

Slouží nám, pro co možná největší snížení zatížení datového spoje a to odstraněním nadbytečné informace (=redundance). Tento druh kódování využívá celou řadu různých algoritmů pro vytváření kódových slov. (Adámek, 1989) Kódy sestavené ze slov stejné délky nazýváme rovnoměrné kódy. Existují také kódy sestavené z kódových slov různé délky, kterým se říká nerovnoměrné kódy. Tyto kódy jsou sestavovány v závislosti na použitém algoritmu a požadovaných vlastnostech.

Bezpečnostní kódování:

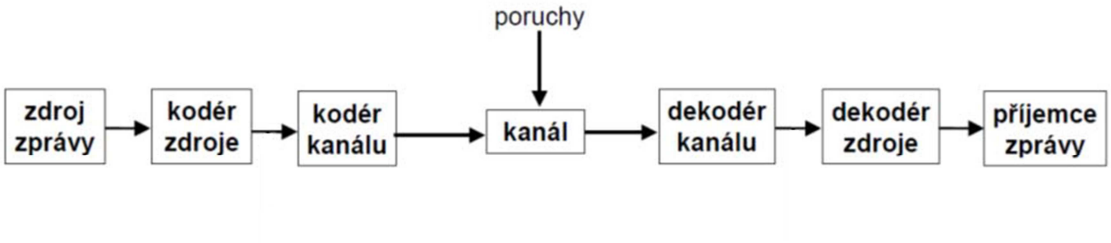
Bezpečnostní kódování se zabývá zabezpečením dat proti chybám, které vznikají na libovolných přenosových cestách, při zápisu (popřípadě čtení) dat z paměťových médií apod. V této práci se nebudeme zabývat příčinami při vzniku chyb.

Díky zakódování informace ovšem zvýšíme velikost přenášených dat. Toto povede k většímu zatížení datové linky. Tím ale získáme možnost, podle výběru konkrétního kódu, k detekci chyb popř. jejich opravu. (Hoffman, a další, 1992)

Šifrování dat:

Tento druh kódování slouží k zabezpečení dat proti odposlouchávání. Nejčastěji ho využijeme na zařízeních komunikujících po éteru. (Costello, a další, 1983)

Pro binárně vyjádřené informace musí vždy platit, že žádné binární slovo nemůže zastupovat dvě různé informace, jinak by docházelo k nejasnostem, při interpretaci této informace.



Obrázek 1 – Schéma komunikačního řetězce

Jak je vidět z tohoto schématu, k dekódování přijatých dat musí docházet v opačném pořadí, než v jakém probíhalo kódování zprávy.

1.2 Bezpečnostní kódování

Bezpečnostní kódování se používá z důvodu zabezpečení informace proti chybám vznikajícím při přenosu z mnoha vlivů působících na přenosovou cestu. Tyto chyby se liší podle použitého druhu přenosové cesty. Při návrhu bezpečnostního kódu pro komunikaci zařízení musíme právě uvažovat o vlastnostech spoje. Bezpečnostní kódy mají totiž tu vlastnost, že čím větší množství chyb dokáží detekovat nebo opravovat, tím jsou tyto kódy delší. Z tohoto důvodu budou při přenosu více zatěžovat síť, popřípadě sníží využitelnou kapacitu přenosového média o právě přidanou nadbytečnou informaci, která slouží k zabezpečení informace. Z těchto důvodu se snažíme volit kompromis mezi robustností kódu, z hlediska zabezpečení zprávy a jeho velikostí.

1.3 Chyby

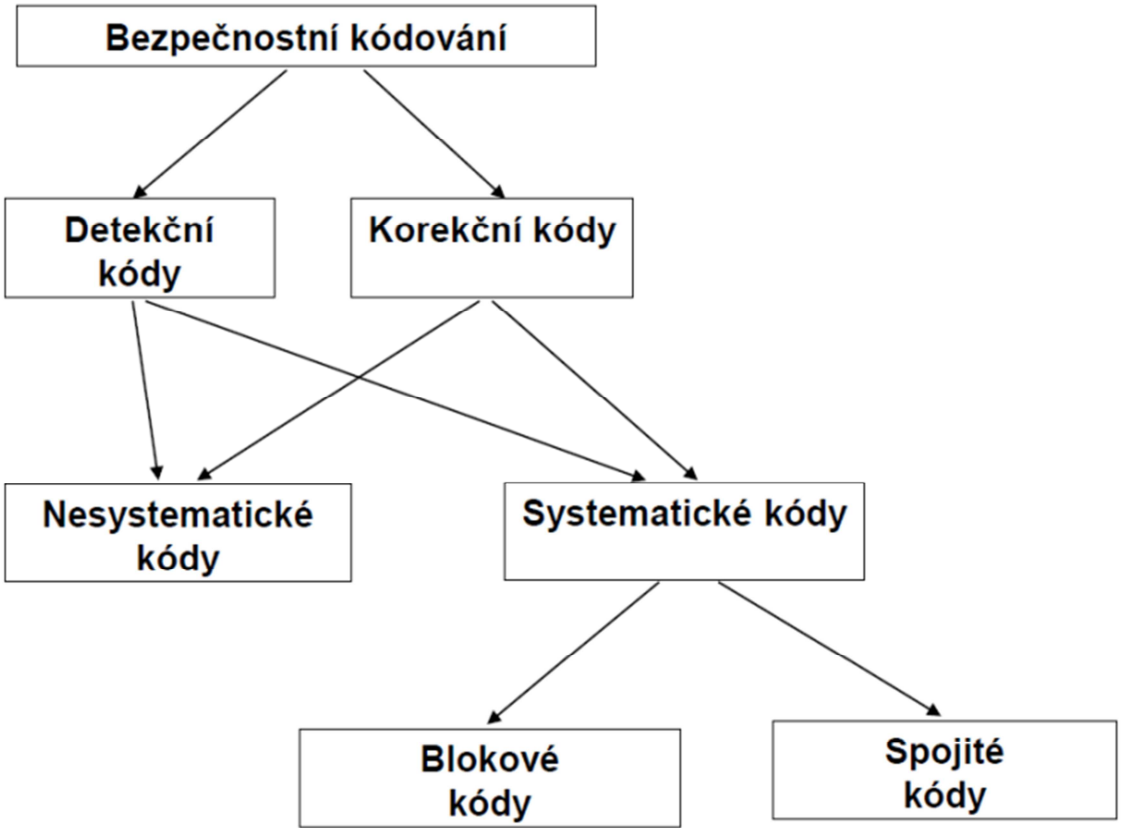
Pro volbu bezpečnostního kódu je také důležité vědět, jaký druh chyb se na přenosové cestě vyskytuje. Tyto druhy chyb jsou úzce spjaty s použitým typem spoje a můžeme je rozdělit do dvou následujících skupin.

- Ojedinělé – „independent error“
  Tyto chyby se nejčastěji vyskytují na metalických vedeních. Tyto chyby se vyskytují samostatně, ovšem může jich být v kódovém slově několik. Proto o nich mluvíme např. jako o t-násobných chybách.
- Shluky – „burst error“
  U toho typu chyb je vždy chybných několik bitů jdoucích po sobě. Tento typ chyb se často vyskytuje u radiového přenosu.

Na každý druh chyb jsou efektivní jiné druhy kódů.

1.4 Druhy bezpečnostních kódů

Bezpečnostní kódování můžeme rozdělit podle následujícího schématu:



Obrázek 2 – rozdělení bezpečnostního kódování

Z obrázku 3 vyplívá, že základním dělením bezpečnostních kódů, je rozdělení na detekční kódy a korekční kódy. Oba tyto druhy kódů se mohou sestavit jako systematické i nesystematické. Následně můžeme ještě systematické kódy rozdělit na blokové a spojité. Jednotlivě pár slov ke každému druhu: (Adámek, 1989)

- **Detekční kódy**

  Tyto kódy jsou sestaveny pouze k informování, zda během přenosu informace nebyla poškozena. Tyto kódy nejsou schopny opravit žádnou chybu. (Hoffman, a další, 1992)
- **Korekční kódy**

  Pokud použijeme tento druh kódů, budeme moci detekovat počet chyb podle vlastností použitého kódu, ale zároveň dostaneme možnost menší počet chyb opravovat. (Hoffman, a další, 1992)

- **Systematické kódy**

  U těchto kódů máme vždy v každém kódovém slově část, která nese informaci a část zabezpečující. Proto tyto kódy označujeme jako (*n*, *k*) kódy, kde *n* je celková délka kódového slova a *k* je počet informačních bitů. (Hoffman, a další, 1992)
- **Nesystematické kódy**

  Tyto kódy jsou sestaveny tak, že v kódovém slově nemáme přesně určeno, jaká část kódového slova je zabezpečovací informace a jaká nese informaci. (Moreira, a další, 2006)
- **Blokové kódy**

  Blokové kódy jsou charakteristické tím, že data jsou kódované po blocích, které mají vždy pro daný kód pevnou délku. (Moreira, a další, 2006)
- **Spojité kódy**

(Moreira, a další, 2006)Ve své práci se budeme zabývat BCH kódy, které patří do skupiny blokových, systematických a korekčních kódů.

1.5 Vybrané pojmy z oboru kódování

Před samotným přistoupením k BCH kódům je vhodné vysvětlit pojmy používané v oboru kódování. Jedná se o:

- Kódové slovo
- Kód
- Hammingova váha
- Hammingova vzdálenost
- minimální vzdálenost kódu.

1.5.1 Kód

Kód je chápán jako soubor všech použitých kódových slov, bude označen jako $K(a)$, kde a je abeceda kódové informace, v našem případě $\{0,1\}$.

1.5.2 Kódové slovo

Kódové slovo je reprezentace zdrojové informace určitou hodnotou, v našem případě binárním slovem. Tato kódová slova můžeme vytvářet pouhým vytvořením tabulky všech možných kódových slov, které můžeme při dané délce slova n vytvořit, a následným přiřazením určit, které kódové slovo reprezentuje kterou informaci. Druhý postup jak zjistit kódové slovo je použít určitý algoritmus na zdrojovou informaci, čímž nám vznikne kódové slovo.

V obou případech je maximální možný počet kombinací kódových slov S v binární soustavě:

$$S = k^n \tag{0}$$

kde k je počet logických úrovní a n je délka kódového slova

V případě binárního kódu je za k dosazeno číslo 2.

1.5.3 Hammingova váha:

Jedná se o číslem vyjádřený počet jedniček v binárním slově (kódu).

Často označováno jako $\mathbf{w}(v)$, kde $\mathbf{v}$ je kódové slovo

Mějme slovo $\mathbf{v} = 01101001$, Hammingova váho tohoto slova bude

$$\mathbf{w}(v) = 4$$

1.5.4 Hammingova vzdálenost:

Pokud budeme uvažovat dvě kódová slova v_1 a v_2, pak nám udává Hammingova vzdálenost počet bitů, ve kterých se tato dvě slova liší. Hammingovu vzdálenost bychom mohli získat například jako Hammingovu váhu výsledku po operaci XOR použitou na tato dvě kódová slova.

$$\mathbf{w}\left(XOR(\mathbf{v}_1, \mathbf{v}_2) \right)$$

Příklad: $\mathbf{v}_1 = 01101001$ a $\mathbf{v}_2 = 10101101$

$$\mathbf{w}\left(XOR\ (\mathbf{v}_1, \mathbf{v}_2) \right) = \mathbf{w}(\ 11000100\ ) = 3$$

1.5.5 Minimální vzdálenost kódu:

Mějme kód o x různých kódových slovech. Poté je minimální vzdálenost kódů, která se většinou značí d_{min} a udává nám nejmenší Hammingovu vzdálenost pro všechny kombinace kódových slov. Tato minimální vzdálenost nám často slouží k porovnávání vlastností různých kódů. Pokud počet všech kombinací, který je odvozen z délky kódového slova n (počet bitů), není plně vytížený, bude Hammingova vzdálenost velmi ovlivněná vhodným rozložením kódových slov.

Pro názornost bychom mohli uvést příklad, kde by byly zakódovány dvě informace na 3 bitech s různými minimálními vzdálenostmi.

možnost: a) ANO = 001 NE = 011

poté je minimální vzdálenost kódu $d_{min} = 1$

b) ANO = 000 NE = 111

minimální vzdálenost $d_{min} = 3$

Zde je vidět, jak vhodným rozdělením můžeme ovlivnit minimální vzdálenost kódu.

Pomocí těchto parametrů můžeme definovat následující vlastnosti kódů, které nám udávají, k čemu můžeme který kód použít.

1.5.6 Detekční schopnost kódu

Je to schopnost kódu, díky níž jsem schopni rozhodnout, zda při přenosu informace došlo někde v kódovém slově k chybě. Počet chyb, které je kód schopný detekovat označme t, potom platí:

$$t \leq d_{\min} - 1 \tag{1}$$

1.5.7 Korekční schopnost kódu

Tuto schopnost definujeme jako schopnost zjistit, na kterých pozicích se vyskytla při přenosu chyba a díky znalosti této pozice chybu opravit. Pokud bychom se pokusili opravit více chyb, než na kolik je kód konstruován, nemůžeme tuto opravu považovat za bezpečnou. Tato schopnost je opět závislá na minimální vzdálenosti kódu a to následovně:

$$t \leq \frac{d_{\min} - 1}{2} \tag{2}$$

Vždy bude pochopitelně platit, že počet opravitelných chyb bude menší než počet detekovatelných chyb.

2 Teoretické podklady lineární algebry

Tyto podklady je nutné ovládat pro pochopení principů a práci celkově v oboru teorie informace. Na principech lineární algebry je založena většina druhů kódování. Nyní se budeme tedy věnovat jednotlivým termínům z oboru lineární algebry, abychom následně mohli přejít k samotnému kódování BCH kódů.

2.1 Tělesa

Definice tělesa:

Těleso *T* je množina se dvěma operacemi (Adámek, 1989) +, ·, takovými, že platí:

1) Operace + vytváří na množině M komutativní grupu s neutrálním prvkem 0

2) Operace · vytváří na množině *M-{0}* všech nenulových prvků komutativní grupu s neutrálním prvkem 1.

3) Distributivní zákon $a(b+c)=ab+ac$ pro libovolné $a,b,c\in M$

Konečné těleso je označení tělesa, které je tvořeno konečným počtem prvků. Každé konečné těleso se také jinak nazývá Galoisovo těleso a značíme jej GF(p^n), kde p je prvočíslo a $n>1$.

Důležitou vlastností je, že Galoisovo těleso $Zp/\mathbf{q}(x)$, nezávisí na výběru konkrétního polynomu $\mathbf{q}(x)$, ale pouze stupně tohoto polynomu.

V této bakalářské práci je použito těleso Z_2 tvořené množinou {0,1}, z důvodu použití pouze binárních čísel.

2.2 Polynomy:

Polynom, také mnohočlen je výraz, který je vyjádřen jako suma násobků mocnin jedné proměnné. Názorně viz následující vzorec:

$$\mathbf{p}(x)=\sum_{i=0}^{n}a_i x^i=a_0+a_1x+a_2x^2+\ldots+a_nx^n \tag{3}$$

Některé ze základních vlastností polynomů:

- Součet polynomů je vždy polynom
- Součin polynomů je vždy polynom
- Derivace polynomu je vždy polynom

Vyjádření binárního čísla pomocí polynomu

V oboru kódování se polynomy používají pro vyjádření samotného binárního čísla. Postup vyjádření binárního čísla jako polynomu je následující. Mocnina x^n je pořadí bitu a koeficient a udává, zda je tento bit v 1 nebo 0. Ukázka vyjádření binárního čísla pomocí polynomu

$$X = 100101$$

$$P(x) = x^5 + x^2 + 1$$

Kořeny polynomu

V oboru polynomů je ještě třeba vysvětlit důležitý pojem kořeny polynomu. Kořenem polynomu rozumíme číslo, označme ho např. α, pro které platí, že dosazením tohoto čísla za proměnnou (v našem případě x) bude výsledkem nula. Matematicky zapsáno:

$$P(\alpha) = 0$$

Ireducibilní polynom

Dále je třeba vysvětlit pojem ireducibilní polynom. Je to takový polynom, který nad daným tělesem Z není tvořen součinem polynomů $a(x)$ a $b(x)$, které jsou polynomy nad tělesem Z nižšího stupně, než je stupeň tohoto polynomu.

Všechny tyto operace s polynomy vyjadřujícími binární čísla budeme muset provádět nad tělesy Z_2, protože binární podoba je z množiny $\{0,1\}$. Proto zde vždy nebudou platit stejná pravidla pro operace s polynomy jako v operacích nad nekonečným tělesem.

Minimální polynom

Minimální polynom je polynom, který v daném tělese pro každý prvek a, který je kořenem některého polynomu s celočíselnými koeficienty, má nejnižší možná stupeň a a je jeho kořenem.

2.3 Algebraické operace nad konečnými tělesy

Jak už bylo řečeno víše, v oblasti kódování pracujeme vždy s konečným tělesem Z_2 a nad tímto tělesem můžeme definovat dvě algebraické operace. Je to operace součtu a operace součinu.

Součet nad tělesem Z_2 můžeme popsat pomocí binární operace nonekvivalence (=XOR) a popsat následující tabulkou:

Tabulka 1 - Součet modulo 2

$\oplus$	0	1
0	0	1
1	1	0

Součin nad tělesem Z_2 můžeme také popsat binární operací a to operací logického součinu (=AND). Tuto operaci bychom mohli popsat následující tabulkou:

Tabulka 2 – Součin modulo 2

$\otimes$	0	1
0	0	0
1	0	1

Obě tyto operace souhrnně nazýváme součet, popřípadě součin modulo-2. To nám značí, že výsledná hodnota, ať už by v běžném součtu nebo násobení vyšla jakákoliv, nikdy nesmí ve výsledku být jiná, než z množiny {0,1}.

3 BCH kódy

3.1 Úvod

BCH kódy patří do skupiny lineárních blokových cyklických kódů. V dnešní době jsou nejčastěji používány při ukládání dat na přenosná média, jako jsou CD, DVD disky nebo u nové technologie blue ray disků. Dále se tyto kódy používají např. v oblasti digitálního satelitního vysílání (standard DVB-S2).

Tyto kódy jsou vystavěny na tzv. Hammingových kódech (Peterson, a další, 1972), ale oproti těmto mají lepší parametry, zejména možnost opravovat větší množství chyb. U BCH kódů máme možnost velké volby jejich parametrů a mají velmi dobrý vztah mezi počtem informačních znaků a počtem opravovaných chyb. Další z jejich předností je, že existuje mnoho dekódovacích technik.

3.2 Zařazení BCH kódů do skupiny

Blokové kódy

Blokové kódy jsou značeny jako (n,k) kódy, kde n je celková délka kódového slova a k je počet informačních znaků. Z tohoto vyplívá, že počet informačních znaků, který budeme značit jako r, je $r = n - k$

Lineární kódy

Lineární kódy jsou založeny na vlastnosti, že každé kódové slovo je lineární kombinací jiných kódových slov. Tyto kódy bývají nejčastěji popsány výše zmiňovanou generující maticí, kde každý řádek tvoří bázi kódu. Generující matice je matice o rozměrech n sloupců a k řádků.

Cyklické kódy

Cyklické kódy jsou množinou kódů, které mají svůj určující prvek tvořen generujícím polynomem. Z tohoto generujícího polynomu vytvoříme generující matici, kde do každého řádku postupně umísťujeme vždy o jednu pozici cyklicky posunutý tento generující polynom.

3.3 Prvky popisující BCH kódy

Generující polynom

Generující polynom je velmi důležitý prvek v oblasti cyklických kódů, který nám slouží k samotnému zakódování vysílaných slov. Můžeme ho označit jako $\mathbf{g}(x)$. Dále je použit, podle konkrétního kódu k vytváření generující matice. V oblasti BCH kódů se generující polynom využívá k vytváření kódových slov přímo jako polynom. U BCH kódů nemáme potřebu pomocí něho sestavovat generující matici.

Matematický zápis generujícího polynomu je následující

$$\mathbf{g}(x) = g_{n-k}x^{n-k} + \ldots + g_2x^2 + g_1x^1 + g_0x^0 \tag{4}$$

Generující matice

V oblasti kódování je generující matice nástroj použitý v lineárních kódech ke generování všech kódových slov v kódu se vyskytujících. Velký význam má generující matice v oblasti cyklických kódů, kde jsou podle této matice zakódována slova. Pro generující matice platí následující pravidla:

- Každý její řádek je tvořen bází, proto musí být vždy kódovým slovem
- Každé nové kódové slovo je lineární kombinací některého řádku
- Každý řádek matice $\mathbf{G}$ je lineárně nezávislý

Pokud zavedeme $\mathbf{b}_1, \mathbf{b}_2, \ldots \mathbf{b}_k$ jako bázi lineárního kódu, bude matematický zápis generující matice $\mathbf{G}$ vypadat následovně.

$$\mathbf{G} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \cdots \\ \mathbf{b}_k \end{pmatrix} \tag{5}$$

Kontrolní matice

Kontrolní matice je většinou vyjádřena pomocí generující matice, to se může u konkrétních kódů lišit, ale u všech typů kódů platí, že je to nástroj pro odhalování chyb a slouží ke kontrole přijatých slov. V této práci tuto matici budeme označovat jako H .

Matematický popis kontrolní matice vypadá následovně.

Slovo $\mathbf{v} = v_1 v_2 v_3 \ldots v_n$ (v T^n) je kódové slovo, pokud platí: $\mathbf{H}\begin{pmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \cdots \\ 0 \end{pmatrix}$,

kde T jsou prvky abecedy kódu.

Kontrolní a generující matice mezi sebou sdílejí následující vztah:

$$\mathbf{G} \times \boldsymbol{H}^T = 0 . \tag{6}$$

3.4 Generující polynom BCH kódů

Výběr a výpočet generujícího polynomu je velmi důležitý pro určení vlastností BCH kódu. Výpočet generujícího polynomu je závislý na výběru kódu. V literatuře lze nalézt tabulky minimálních vzdáleností, které jsou podle výše uvedených popisů vlastností potřebné pro vlastnosti kódu, pro jednotlivé kombinace n a k bitů. Pro názornost zde bude uvedena tabulka kódů délky $n = 7,15,31$.

Tabulka 3 - Vybrané (n,k) kódy a kódové vzdálenosti

Délka (n)	Minimální vzdálenost (d_{min})	Informační znaky (k)
7	3	4
	7	1
	3	11
15	5	7
	7	5
	15	1
31	3	26
	5	21
	7	16
	11	11
	15	6
	31	1

Generující polynom získáme pomocí minimálních polynomů daného Galoisova tělesa. Označme tyto minimální polynomy jako m. Potom pro získání generujícího polynomu nám stačí vybrat jeden z těchto minimálních polynomů. Ovšem podle Boseho a Chaudhuriho teorému uvedeného níže, podle kterého jsme schopni získat minimální kódovou vzdálenost výsledného kódu, není takto vybraný generující polynom vždy dostačující.

Boseho a Chaudhuriho teorém:

Jestliže se mezi kořeny vytvářecího mnohočlenu cyklického (n, k) kódu nachází určitý počet prvků Galoisova tělesa jdoucích velikostí svých exponentů po sobě, což zapíšeme takto: a^m, a^{m+1}, ... , a^{m+d-2} , pak minimální Hammingova vzdálenost d_{min} tohoto kódu není menší než d . (Němec, 2005)

Kořeny minimálních polynomů každého GF se dají snadno dopočítat dosazováním postupně jednotlivých prvků GF, ale dají se také dohledat tabulky těchto kořenů. Pro názornost zde bude uvedena tabulka kořenů pro minimální polynomy tělesa GF(16)

Tabulka 4 – Minimální polynomy GF(24)

i	Minimální polynom	Kořeny polynomu
1	$x^4 + x^1 + 1$	a^1, a^2, a^4, a^8
2	$x^4 + x^3 + x^2 + x^1 + 1$	a^3, a^6, a^9, a^{12}
3	$x^2 + x^1 + 1$	a^5, a^{10}
4	$x^4 + x^3 + 1$	$a^7, a^{11}, a^{13}, a^{14}$

Odtud je vidět, že podle BC teorému by při zvolení generujícího polynomu jako $\mathbf{m}_1(x)$ by byla minimální vzdálenost rovna $d_{min} = 3$ a proto podle výše uvedených pravidel tento kód dokáže opravit pouze jednu chybu. Abychom získali lepší vlastnosti, musíme zvolit generující polynom jako součin několika minimálních polynomů, tak abychom podle BC teorému získali větší kódovou vzdálenost.

Vybraný generující polynom tedy získáme kombinací minimálních polynomů:

$$\mathbf{g}(x) = \mathbf{m}_1(x) \bullet \mathbf{m}_2(x) \bullet \mathbf{m}_3(x) ... \tag{7}$$

3.5 Kódování BCH

Kódování probíhá jako u cyklických kódů, a to následovně:

- Cyklickým posuvem informačního slova o *n-k* znaků
- Vypočítání zbytku po dělení posunutého informačního slova generujícím polynomem
- Sečtení posunutého informačního slova se zbytkem po dělení z předchozího kroku

V polynomiálním vyjádření se cyklický posuv provádí pomocí vynásobení polynomu informačního slova polynomem x^{n-k} . Informační slovo označme $\mathbf{z}(x)$ Tím nám vznikne slovo požadované délky n.

$$\mathbf{z}'(\mathrm{x}) = \mathbf{z}(x) \bullet x^{n-k} \tag{8}$$

Toto prodloužené informační slovo vydělíme generujícím polynomem a vzniknou nám dva polynomy, které označíme jako $\mathbf{M}(x)$ a $\mathbf{R}(x)$. Pro tuto práci zavedeme $\mathbf{M}(x)$ jako podíl dělení a $\mathbf{R}(x)$ jako zbytek po dělení. Potom máme na výběr dvě možnosti jak získat kódové slovo.

- Sečtením $\mathbf{z}(x)$ a $\mathbf{R}(x)$
- Součinem $\mathbf{M}(x)$ a $\mathbf{g}(x)$

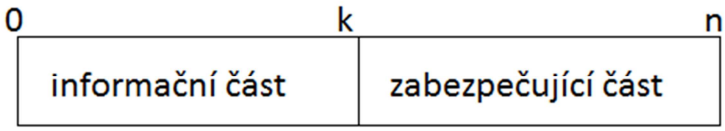
Proto tedy vzorec pro získání kódového slova vypadá následovně:

$$\frac{\mathbf{z}(x) \bullet x^{n-k}}{\mathbf{g}(x)} = \mathbf{M}(x) + \frac{\mathbf{R}(x)}{\mathbf{g}(x)} \tag{9}$$

Pokud tento vztah upravíme vynásobením $\mathbf{g}(x)$ a přičtením $\mathbf{r}(x)$ dostaneme výsledný tvar, ze kterého je vidět rovnost mezi oběma způsoby zjištění kódového slova:

$$\mathbf{z}(x) = \mathbf{z}'(x) \bullet x^{n-k} + \mathbf{r}(x) = \mathbf{M}(x) \bullet \mathbf{g}(x) + \mathbf{R}(x) + \mathbf{R}(x) = \mathbf{M}(x) \bullet \mathbf{g}(x) \tag{10}$$

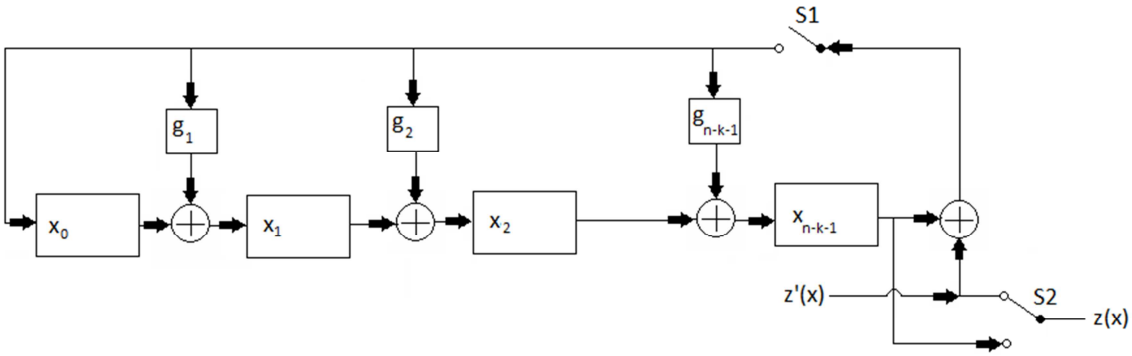
Výsledné kódové slovo tedy bude mít tvar blokového kódu a to:



Obrázek 3 - Schéma kódového slova BCH kódu

3.5.1 Realizace kodéru BCH kódů

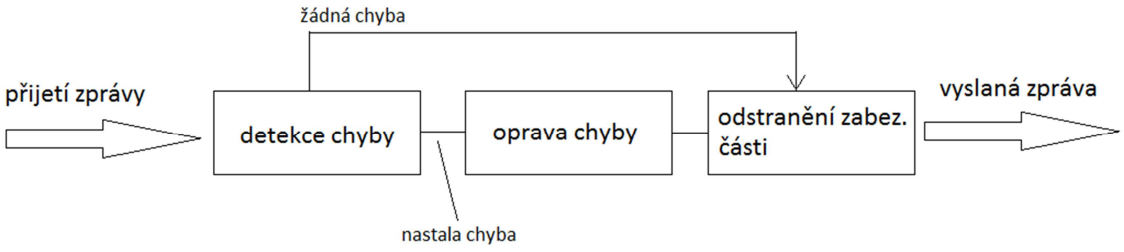
Kodér BCH kódů se v praxi realizuje pomocí posuvného registru a sčítaček modulo 2. Posuvný registr musí obsahovat paměťových buněk tolik, kolik je zabezpečovacích bitů a to $(n-k)$. Jak je vidět z obrázku (obrázek 5) uvedeného níže, podle vytvářecího polynomu je zavedena sčítačka modulo 2, stejně jako zpětné vazby pro posuvný registr. Zapojení je odpovídající pro cyklické kódy a je snadno dohledatelné ve většině publikací, zabývajících se kódováním. (Číka, 2006)



Obrázek 4 – kodér BCH kódů

3.6 Dekódování BCH kódu

Dekódování spočívá ve zjištění, pomocí algoritmu, zda se během přenosu nebo ukládání informace nevyskytla chyba, která by zapříčinila poškození této informace. Pokud bychom nezjistili žádnou chybu, není vždy zaručené, že přijatá informace bude totožná s vyslanou. Mohla by totiž nastat situace, kdy došlo k více chybám, než na kolik byl kód konstruován. Pokud ovšem opravdu k žádné chybě nedošlo, bude přijatá informace totožná s vyslanou a následně nám stačí odstranit zabezpečovací část, čímž získáme požadovanou zdrojovou informaci. Nyní ještě zbývá varianta, kdy nastal počet chyb, který je kód schopen detekovat. Pokud je kód schopen opravit počet vzniklých chyb, dojde k jejich korekci. Tato korekce se provádí různými algoritmy, o kterých bude řeč dále. My v tomto případě opět zjistíme původní vyslanou informaci. V případě, že kód není konstruován na dostatečný počet chyb, zjistíme pouze, že chyba nastala, ale nemůžeme ji opravit. V takovém případě si musíme informaci vyžádat znovu, je-li toto možné. Poslední variantou je možnost, kdy se vyskytne více chyb, než je detekční schopnost kódu. V takovém případě chybu ani nezjistíme a pouze dekódujeme zprávu. Tato zpráva sice odpovídá, některé z možných zpráv, ovšem neodpovídá vyslané. Schéma dekódování je následující:



Obrázek 5 – schéma dekódování

BCH kódy mají velmi propracované dekódovací techniky, a proto jsou v dnešní době velmi využívané. Všechny techniky jsou založeny na následujícím postupu:

(1) Vypočtení syndromů
(2) Zjištění počtu chyb
(3) Výpočet lokátoru chyb
(4) Výpočet chybového polynomu
(5) Opravení chyb

Ještě je nutno říci, že existuje velmi mnoho algoritmů pro dekódování. V této práci se budu zabývat pouze Petersonovým algoritmem. Všechny ostatní metody jsou velmi snadno dohledatelné v publikacích zabývajícími se teorií BCH kódů. Tyto metody jsou:

- Petersonův algoritmus (Němec, 2005)
- Berlekamp-Massey algoritmus (Costello, a další, 1983)
- Euklidův algoritmus (Moon, 2005)
- Maticová metoda (Adámek, 1989)

3.6.1 Výpočet syndromů

Výpočet syndromů spočívá v dosazení kořenů generujícího polynomu do polynomu přijatého slova. O těchto kořenech a detekčních schopnostech BCH kódu je více informací v kapitole 4.3 (Němec, 2005). Po dosazení těchto kořenů zjistíme soustavu syndromů, které budeme v následujícím kroku používat ke zjištění lokátoru chyb. Počet syndromů bude shodný s počtem kořenů použitého generujícího polynomu.

Zde mohou nastat dva případy. Pokud všechny syndromy vyjdou nulové, je možné předpokládat, že nenastala chyba a proto můžeme přejít přímo k odstranění nadbytečné informace. V opačném případě přistoupíme k dalšímu kroku.

3.6.2 Zjištění počtu chyb

Tento krok patří k nejobtížnější části dekódování. Zde je třeba převést soustavu syndromů, které jsme vypočítali v předchozím kroku do soustavy vazebních rovnic. Těchto rovnic bude tolik, kolik chyb je kód schopen opravit. Tuto korekční schopnost kódu zde označme jako *v*.

Potom bude soustava vypadat následovně

$$\begin{aligned} S_1 z_v + S_{1+1} z_{v-1} + \ldots + S_{1+(v-1)} z_1 + S_{1+v} &= 0 \\ S_2 z_v + S_{2+1} z_{v-1} + \ldots + S_{2+(v-1)} z_1 + S_{2+v} &= 0 \\ S_v z_v + S_{v+1} z_{v-1} + \ldots + S_{v+(v-1)} z_1 + S_{v+v} &= 0 \end{aligned} \qquad (11)$$

V této soustavě je potřeba určit počet lineárně nezávislých rovnic. Tomuto počtu se říká hodnost matice, označme ji h. Tehdy, bude počet ve skutečnosti vzniklých chyb roven právě této hodnosti. Potom pro výpočet lokátoru chyby budeme pracovat s h rovnicemi.

3.6.3 Výpočet lokátoru chyby

Zjištění soustavy rovnic pro výpočet lokátorů chyb je popsáno v předchozí kapitole. Podle hodnosti matice zjistíme, kolik ve skutečnosti nastalo chyb. Proto se tato soustava zmenší na $v-h$ rovnic, ale také se bude hledat pouze h lokátorů chyb. Postup pro odstranění nadbytečných lokátorů a rovnic je jednoduchý. Všechny lokátory, které mají index větší, než h se položí rovny nule. $Z_{v>h} = 0$. A použijeme pouze prvních h rovnic.

Tyto lokátory chyb jsou potřeba do dalšího kroku.

3.6.4 Výpočet chybového vektoru

Z předchozího kroku byly zjištěny lokátory chyb. Tyto lokátory musíme dosadit do rovnice následující rovnice:

$$X^h + X^{h-1} z_1 + ... + z_h = 0 \tag{12}$$

Po dosazení vypočítaných lokátorů chyb je nutné zjistit kořeny této rovnice ovšem nad zvoleným tělesem GF. Pokud budou všechny tyto kořeny známy, jejich exponenty udávají pozice chyb v přijatém slově.

Chybový vektor bude značen jako $\mathbf{e}(x)$. Pokud bychom získali následující kořeny rovnice:

$$a^{i_1}, a^{i_2}, ..., a^{i_h}$$

získáme chybový vektor:

$$\mathbf{e}(x) = e^{i_1}, e^{i_2}, ..., e^{i_h} . \tag{13}$$

Odstranění chyby

Odstranění chyby je jednoduchý proces. Protože chyba je způsobena právě přičtením chybového vektoru k vyslanému slovu, vyslané slovo se zjistí odečtením tohoto chybového vektoru od přijatého slova. V tělesech Z_2 je ovšem proces rozdílu stejný jako součtu. Proto se k přijatému slovu přičte chybový vektor a výsledkem bude vyslané slovo. V zápisu polynomů bude toto vypadat následovně:

$$\mathbf{v}(x) = \mathbf{w}(x) + \mathbf{e}(x), \tag{14}$$

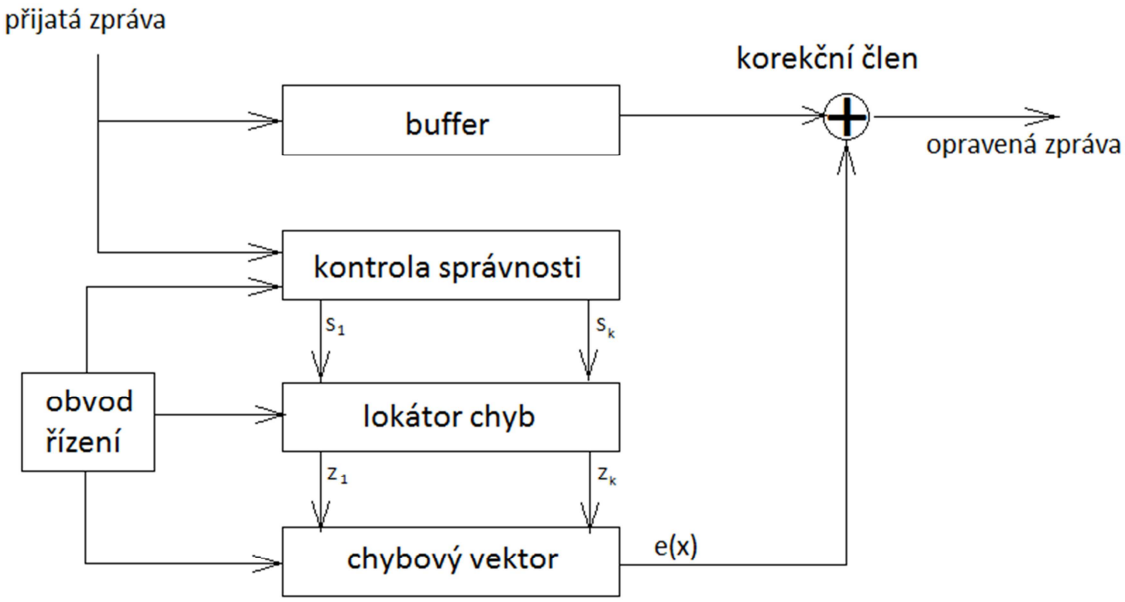
kde $\mathbf{v}(x)$ je polynom vyslaného slova a $\mathbf{w}(x)$ je polynom přijatého slova.

3.6.5 Zjištění vyslaného slova

V posledním kroku zjištění vyslaného slova je třeba odstranit zabezpečovací složku kódu. U blokových kódů je tento proces velice jednoduchý, protože se jedná pouze o součet dvou polynomů. Proto ke zjištění jednoho z nich postačí pouze uvažovat blok kódu, ve které se nachází informační část. Pokud by informace byla v hardwarové podobě uložena v registru délky n bitů, stačí vybrat pouze k nejvyšších bitů. V případě polynomu je třeba pouze vydělit tento polynom prvkem s exponentem o velikosti $n-k$.

3.6.6 Realizace dekodéru BCH

Protože realizace dekodéru pomocí elektronických prvků by byla velmi složitá, a v této práci není reálný dekodér předmětem práce, bude zde uvedeno pouze principiální schéma.



Obrázek 6 – schéma BCH dekódování

4 Ukázka výpočtu

V této kapitole bude ukázán typický postup kódování a dekódování zprávy s výpočtem jednotlivých kroků. Zadání je následující

- Počet informačních bitů 10 [bit]na přenosový kanál
- Celková délka kódu n – 15 [bit]
- Maximální počet chyb 2
- Vstupní zpráva $v = 1010101010$

Nyní přejdeme k jednotlivým krokům.

4.1 Kódování zprávy

Výběr a výpočet generujícího polynomu

Podle zadání nejprve musíme vybrat parametr kódu *n* tak, aby odpovídal požadavkům na přenosový kanál. V našem případě se jedná o 15 bitů. Tudíž parametr $n = 15$. Od tohoto se odvíjí i výběr Galoisova tělesa. Těleso vybereme tak, aby GF(X) odpovídalo $X = n + 1$.
V našem případě tedy vybrané těleso bude GF(16). Zbytky po dělení pro jednotlivé kořeny α v tělese GF(16) jsou uvedeny v příloze A.

Další předpoklad je, že číslo 16 můžeme vyjádřit buď jako $16=2^4$ nebo jako $16=4^2$.
V kapitole 3 jsme řekli, že Galoisovo těleso je těleso Z_p, kde p musí být prvočíslo, tudíž pro náš případ odpovídá $16=2^4$.

Nyní již máme vybrané těleso a zbývá nám vybrat parametr kódu k. Tento parametr určuje, jaký bude poměr informační části a zabezpečující části a tudíž i vlastnosti kódu po stránce počtu opravitelných chyb. Protože od kódu vyžadujeme, aby opravil 2 chyby, musíme zjistit požadovanou kódovou vzdálenost (3). Ta je určujícím parametrem počtu opravitelných chyb kódu. K tomuto číslu se dopočítáme následovně

$$t \le \frac{d_{\min} - 1}{2} \quad \Rightarrow \quad d_{\min} \ge 2t + 1$$

a protože $t = 2$, dostaneme výsledek $d_{\min} = 5$ a proto vybereme z tabulky 3 délku informační části $k = 7$.

Dále je nutné určit generující polynom pro vybraný kód. V tabulce 4 jsou ukázány minimální polynomy, které budou sloužit k výpočtu výsledného generujícího polynomu. Podle BC teorému (viz. kapitola 4), vybereme minimální polynomy podle jejich kořenů tak, aby exponenty těchto kořenů byli po sobě jdoucí až do exponentu 4. $\exp onent = m + d_{\min} - 2 = 1 + 5 - 2 = 4$. Tudíž z tabulky 3 vybereme minimální polynomy

$\mathbf{m}_1(x)$ a $\mathbf{m}_2(x)$, které budou mít kořeny α^1, α^2, α^3, α^4, α^6, α^8, α^9, α^{12}. Poté použijeme (odkaz na vzorec) a dostaneme tento generující polynom.

$$\mathbf{g}(x) = \mathbf{m}_1(x) \cdot \mathbf{m}_2(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) = \mathrm{x}^8 + \mathrm{x}^7 + \mathrm{x}^6 + \mathrm{x}^4 + \mathrm{x} + 1.$$

V tuto chvíli máme všechny potřebné podklady pro přistoupení k dalšímu kroku a zakódování zprávy.

Kódování zprávy

Ještě než přistoupíme k samotnému zakódování je potřeba zavést zprávu na vstupu do jednotlivých bloků. Tyto bloky jsou ukázány níže. Délku těchto bloků určuje právě parametr *k* vybraného kódu. Protože jsme zvolili $k = 7$, musíme zprávu rozdělit do bloků o délce 7 b. Pokud by po rozdělení zbylo několik bitů z původní zprávy, musíme tento zbytek doplnit např. nulami na požadovanou délku. V našem případě máme zprávu na vstupu dlouhou 10b a potřebujeme bloky 7b, získáme 2 bloky, kde první bude délky 7b ze zprávy na vstupu a v druhém bloku budou 3b, které zbyly a doplníme 4 bity nulami. Tudíž dostaneme dva bloky zprávy.

v_1=1010101

v_2=010**0000**

Nyní se budeme věnovat práci pouze s prvním blokem pro zjednodušení tohoto popisu.

V kapitole 4 o BCH kódech je princip kódování a dekódování obecně popsán, proto se zde již nebudu zabývat detailním popisem. Jako první krok převedeme zprávu z binární podoby do podoby polynomiální. Slovo $\mathbf{v}_1$ vyjádříme následovně $\mathbf{v}_1(x) = x^6 + x^4 + x^2 + 1$. Před dělením informačního bloku generujícím polynomem, je třeba tento blok rozšířit vynásobením x^{n-k} . Rozšířené informační slovo bude následující:

$$\mathbf{z}_1'(x) = \mathbf{v}_1(x) \cdot (x^{n-k}) = (x^6 + x^4 + x^2 + 1) \cdot x^8 = x^{14} + x^{12} + x^{10} + x^8$$

Toto rozšířené informační slovo následně vydělíme generujícím polynomem, kde výsledkem bude požadovaný zbytek $\mathbf{R}(x)$.

$$\mathbf{M}(x) = \mathbf{z}_1'(x) : \mathbf{g}(\mathrm{x}) =$$
$$(x^{14} + x^{12} + x^{10} + x^8) : (\mathrm{x}^8 + \mathrm{x}^7 + \mathrm{x}^6 + \mathrm{x}^4 + \mathrm{x} + 1) = \mathrm{x}^6 + \mathrm{x}^5 + \mathrm{x}^4 + \mathrm{x}^2 + 1$$
$$\underline{-(\mathrm{x}^{14} + \mathrm{x}^{13} + \mathrm{x}^{12} + \mathrm{x}^{10} + \mathrm{x}^7 + \mathrm{x}^6)}$$
$$(\mathrm{x}^{13} + \mathrm{x}^8 + \mathrm{x}^7 + \mathrm{x}^6)$$
$$\underline{-(\mathrm{x}^{13} + \mathrm{x}^{12} + \mathrm{x}^{11} + \mathrm{x}^9 + \mathrm{x}^6 + \mathrm{x}^5)}$$
$$(\mathrm{x}^{12} + \mathrm{x}^{11} + \mathrm{x}^9 + \mathrm{x}^8 + \mathrm{x}^7 + \mathrm{x}^5)$$
$$\underline{-(\mathrm{x}^{12} + \mathrm{x}^{11} + \mathrm{x}^{10} + \mathrm{x}^8 + \mathrm{x}^5 + \mathrm{x}^4)}$$
$$(\mathrm{x}^{10} + \mathrm{x}^9 + \mathrm{x}^7 + \mathrm{x}^4)$$
$$\underline{-(\mathrm{x}^{10} + \mathrm{x}^9 + \mathrm{x}^8 + \mathrm{x}^6 + \mathrm{x}^3 + \mathrm{x}^2)}$$
$$(\mathrm{x}^8 + \mathrm{x}^7 + \mathrm{x}^6 + \mathrm{x}^4 + \mathrm{x}^3 + \mathrm{x}^2)$$
$$\underline{-(\mathrm{x}^8 + \mathrm{x}^7 + \mathrm{x}^6 + \mathrm{x}^4 + \mathrm{x} + 1)}$$
$$(\mathrm{x}^3 + \mathrm{x}^2 + \mathrm{x} + 1) \quad = \mathrm{R}(\mathrm{x})$$

Tímto jsme získali zbytek po dělení $\mathbf{R}(x)$ a ten nyní podle vzorce 8 pouze přičteme ke slovu $\mathbf{z}(x)$ a tím získáme zakódované slovo.

$$\mathbf{z}_1(x) = \mathbf{z}_1'(x) + \mathbf{R}(x) = (x^{14} + x^{12} + x^{10} + x^8) + (x^3 + x^2 + x + 1) =$$
$$= (x^{14} + x^{12} + x^{10} + x^8 + x^3 + x^2 + x + 1)$$

V binárním vyjádření: $z_1 = 1010101000011111$

Takto zakódované slovo je připraveno pro zpracování. Nyní přejděme k zavedení chyby.

Zavedení chyby

Zavedení chyby je pouze jednoduché přičtení modulo-2 polynomu, který má prvky pouze na pozicích bitů, kde nastala chyba. Náhodně zvolíme např. 2 chyby na místech 12. a 5. bitu. Z toho vyplívá, že chybový vektor je $\mathbf{e}(x) = \mathrm{x}^{12} + \mathrm{x}^5$. Proto přijaté slovo s chybou bude následující.

$$\mathbf{w}_1(x) = \mathbf{z}_1(\mathrm{x}) + \mathbf{e}(\mathrm{x}) = (x^{14} + x^{12} + x^{10} + x^8 + x^3 + x^2 + x + 1) + (\mathrm{x}^{12} + \mathrm{x}^5) =$$
$$= x^{14} + x^{10} + x^8 + x^5 + x^3 + x^2 + x + 1$$

Toto kódové slovo s chybou nyní budeme uvažovat při procesu dekódování.

4.2 Dekódování zprávy

Jako první je třeba zjistit, zda vůbec k nějaké chybě při zpracování zprávy došlo. Toto zjistíme pomocí výpočtu syndromů, které nám budou následně sloužit k dalšímu výpočtu v případě, že by chyba nastala.

Výpočet syndromů

V našem případě budeme zjišťovat 4 syndromy, protože náš generující polynom je tvořen minimálními polynomy, které v součinu mají 4 kořeny s po sobě jdoucími exponenty. Zde už bude využita tabulka uvedena v příloze A Galoisovo těleso GF(16). Konkrétně k výpočtu jednotlivých syndromů.

S_1:

$$\mathbf{w}_1(\alpha^1) = (\alpha^{14} + \alpha^{10} + \alpha^8 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha + 1)$$
$$= (\alpha^3 + 1 + \alpha^2 + \alpha + 1 + \alpha^2 + 1 + \alpha^2 + \alpha + \alpha^3 + \alpha^2 + \alpha + 1)$$
$$= \alpha$$

S_2:

$$\mathbf{w}_1(\alpha^2) = (\alpha^{28} + \alpha^{20} + \alpha^{16} + \alpha^{10} + \alpha^6 + \alpha^4 + \alpha^2 + 1)$$
$$= (\alpha^{13} + \alpha^5 + \alpha + \alpha^{10} + \alpha^6 + \alpha^4 + \alpha^2 + 1)$$
$$= (\alpha^3 + \alpha^2 + 1 + \alpha^2 + \alpha + \alpha + \alpha^2 + \alpha + 1 + \alpha^3 + \alpha^2 + \alpha + 1 + \alpha^2 + 1)$$
$$= \alpha^2$$

S_3:

$$\mathbf{w}_1(\alpha^3) = (\alpha^{42} + \alpha^{30} + \alpha^{24} + \alpha^{15} + \alpha^9 + \alpha^6 + \alpha^3 + 1)$$
$$= (\alpha^{12} + \alpha^0 + \alpha^9 + \alpha^0 + \alpha^9 + \alpha^6 + \alpha^3 + 1)$$
$$= (\alpha^3 + \alpha^2 + \alpha + 1 + \alpha^3 + \alpha^2 + \alpha^3 + 1)$$
$$= \alpha^3 + \alpha = \alpha^9$$

S_4:

$$\mathbf{w}_1(\alpha^4) = (\alpha^{56} + \alpha^{40} + \alpha^{32} + \alpha^{20} + \alpha^{12} + \alpha^8 + \alpha^4 + 1)$$
$$= (\alpha^{11} + \alpha^{10} + \alpha^2 + \alpha^5 + \alpha^{12} + \alpha^8 + \alpha^4 + 1)$$
$$= (\alpha^3 + \alpha^2 + \alpha + \alpha^2 + \alpha + 1 + \alpha^2 + \alpha^2 + \alpha + \alpha^3 + \alpha^2 + \alpha + 1 + \alpha^2 + 1 + \alpha + 1 + 1)$$
$$= \alpha + 1 = \alpha^4$$

Z nenulovosti syndromů vidíme, že v přijatém slově w došlo k chybě (chybám). Nyní zbývá určit počet a pozice chyb, k čemuž slouží tzv. lokátory chyb.

Výpočet lokátoru chyb

Z vypočítaných syndromů nejprve podle vzorce 9 z kapitoly 4 musíme sestavit soustavu rovnic pro výpočet lokátoru chyb. Korekční schopnost kódu jsme zjistili, je 2 chyby, a proto $v = 2$. Výsledná soustava vypadá následovně.

$$\begin{aligned} &S_1 z_2 + S_2 z_1 + S_3 = 0 \\ &S_2 z_2 + S_3 z_1 + S_4 = 0 \\ &\alpha z_2 + \alpha^2 z_1 + \alpha^9 = 0 \\ &\alpha^2 z_2 + \alpha^9 z_1 + \alpha^4 = 0 \end{aligned}$$

Nejprve je nutné zjistit počet lineárně závislých rovnic. Tím zjistíme, kolik chyb skutečně nastalo a podle toho následně upravíme tuto soustavu. Tato úprava je vysvětlena v kapitole 3. V našem případě je počet lineárně nezávislých rovnic roven 2, což vede ke zjištění skutečnosti, že nastaly 2 chyby. Pro ukázku odstranění nadbytečných rovnic řekněme, že by byla hodnost matice 1 a proto bychom museli odstranit jednu rovnici. Podle popisu v kapitole 3 by pak výsledná soustava vypadala následovně:

$$\alpha^2 z_1 + \alpha^9 = 0$$

Zde je vidět, že jsme odstranili poslední jednu rovnici a také jednu neznámou Z a to konkrétně Z_2 .

V dalším kroku dopočítáme lokátory chyb z_1 a z_2 pomocí pravidel pro výpočet soustavy lineárních rovnic v oboru GF. $z_1 = \alpha^{14} \quad z_2 = \alpha^2$.

Tyto lokátory nám tedy budou sloužit pro výpočet vektoru chyb.

Výpočet chybového vektoru

Nyní již pouze zbývá sestavit rovnici vektoru chyb pomocí lokátorů chyb.

$$X^2 + X z_1 + z_2 = X^2 + X\alpha^{14} + \alpha^2 = 0$$

V této chvíli stačí pouze zjistit kořeny této rovnice, kde exponenty těchto kořenů nám udávají pozice vzniklých chyb. V našem případě to jsou α^5, α^{12} .

Opravení chyby

Nyní už zbývá sestavit chybový vektor, který se přičte modulo-2 k vektoru přijaté zprávy a tím získáme opravenou přijatou zprávu.

$$\mathbf{v}_1(x) = \mathbf{w}_1(x) + \mathbf{e}_1(x) = (x^{14} + x^{10} + x^8 + x^5 + x^3 + x^2 + x + 1) + (x^{12} + x^5) =$$
$$= x^{14} + x^{12} + x^{10} + x^8 + x^3 + x^2 + x + 1$$

Odtud je vidět, že odpravené slovo odpovídá přijatému.

Sestavení přijaté zprávy

Posledním krokem je sestavit z jednotlivých bloků přijatou zprávu tak, aby korespondovala s vyslanou. Toto spočívá ve vybrání informační části přijatého a opraveného kódového slova, což je prvních k bitů. Pro názornost v uvedeném příkladu se nejprve převede přijaté slovo z polynomiální podoby do podoby binárního slova.

$$\mathbf{v}_1(x) = x^{14} + x^{12} + x^{10} + x^8 + x^3 + x^2 + x + 1 \Rightarrow v_1 = 101010100001111$$

Odtud tedy vybereme prvních 7 bitů, což jsou $v_1' = 1010101$. Nyní uvažujme, že ve slově v_2 nenastala žádná chyba a postupovali bychom obdobně, takže bychom dostali $v_2' = 0100000$. Takto přijatá slova musíme opět sloučit do zprávy délky 10b. To dosáhneme způsobem, kdy všechna tato slova sloučíme postupně za sebe a budeme vybírat vždy po 10 bitech. Přebývající bity na konci takto sloučené zprávy jsou doplněné a tudíž pro nás nedůležité a proto je nemusíme dále uvažovat. Tento postup bude vypadat následovně.

$$v' = v_1', v_2' = 10101010100000$$

Odtud vybereme prvních 10b.

$$v = 1010101010$$

Celý proces proběhl úspěšně. Dekódovali jsme i přes vzniklé chyby stejné slovo jako slovo, které bylo vysláno.

5 Rozbor aplikace

V této poslední kapitole bude rozbor vytvořené aplikace, která byla předmětem této bakalářské práce. Tato aplikace byla vytvořena jako ilustrační prvek při výuce teorie informace. Aplikace postupně ukazuje jednotlivé kroky, které jsou provedeny během procesu kódování a dekódování současně se zavedením chyby v průběhu zpracování.

Tato aplikace byla vytvořena ve vývojovém prostředí Matlab R2009. (The MathWorks, Inc., 1994) Matlab patří k nejrozšířenějším matematickým softwarům a je založen na maticových výpočtech. (Zaplatílek, a další, 2003)

Nyní bude ukázáno prostředí aplikace s konkrétním příkladem, kde vstupující zpráva pro kodér bude věta: „zkusebni text“. Vybraný kód bude BCH(15,5).

Obrázek 7 - uživatelské prostředí aplikace

Bod1 – Pole vstupního textu

Toto pole slouží k zadání vstupního textu, který uživatel této aplikace chce zakódovat. Uživatel má dvě možnosti, jak toto pole vyplnit. První způsob, je manuálním zápisem požadované věty. Druhý způsob, jak zadat vstupní text je pomocí funkce import. Tato funkce je k nalezení v menu – možnosti – import. (Bod 9) Tato aplikace je vytvořena pro práci bez diakritiky a proto je doporučeno nezadávat jako vstupní text znaky s diakritikou.

Bod2 – Pole výběru kódu

Pomocí těchto dvou roletových panelů si uživatel vybírá, jaký typ BCH kódu bude použit pro zakódování zprávy. Protože pro různé délky kódu n, jsou jiné délky k je vždy při výběru n změněn obsah rozbalovacího panelu tak, aby odpovídal možnostem pro vybrané *n*. Tyto kombinace jsou k nahlédnutí v kapitole 4 v tabulce 3. Pokud uživatel již pomocí jednoho kódu kódoval, při výběru jiného kódu budou všechna textová pole v oblastech „Kódování“, „Zavedení chyby“ a „Dekódování“ vymazána.

- **Bod3 – Pole parametrů kódu**
  Tato pole nemá uživatel aplikace možnost přímo změnit. Slouží pouze k ukázce určitých parametrů kódu, který byl vybrán. (Bod 2) V tomto bodě uživatel zjistí, jaký generující polynom $\mathbf{g}(x)$ je použit pro kódování, jakou korekční schopnost vybraný kód bude mít a také Hammingovu vzdálenost vybraného kódu.
- **Bod4 – Pole reset**
  Tlačítko [Reset] slouží k vymazání obsahu a zablokování tlačítek pro uložení jednotlivých kroků a tlačítek [Zaveď chybu] a [Dekóduj]. Funkce, která ovládá toto tlačítko, je volána právě při změně hodnoty z rozbalovacích oken (Bod 2), pro výběr jiného kódu.
- **Bod5 – Pole oblasti kódování**
  Tato oblast aplikace slouží k prezentaci jednotlivých kroků kódování. První textové okno „Zpráva v ascii“ uvádí převedení jednotlivých znaků do ascii podoby. Textové pole „Převedení do bloků“ ukazuje, jak se ascii hodnoty poskládají za sebe do jednoho řádku a následně po krocích vyberou a řadí po řádcích do bloků. Třetí okno „Zakódované zprávy“ ukazuje, jak vypadá zakódovaná zpráva pomocí generujícího polynomu $\mathbf{g}(x)$ (Bod 3). Takto zakódovaná zpráva bude délky n.

  Tato pole se vyplní po stisknutí tlačítka [Zakóduj]. Toto tlačítko zároveň zpřístupní tlačítka [Zaveď chybu] a [Ulož] v oblasti Kódování.
  Tlačítko ulož, slouží k výběru souboru, do kterého má být oblast kódování uložena. Pomocí toho tlačítka budou uloženy následující pole: Generující polynom, zpráva v ascii, zpráva převedená do bloků a zakódovaná zpráva.

- **Bod6 – Pole oblasti zavedení chyby**

  Pole pro zavedení chyby obsahuje opět 3 okna, kde první slouží uživateli pro zadávání chyb, a zbylé dvě jako informační bloky. První textové pole je vyplněno stejným obsahem, jako pole „Zakódované zprávy“ (Bod 5) a uživatel ho má možnost editovat. Uživatel má pomocí výběrových prvků, umístěných nad textovým polem „Zavedení chyby“, možnost vybrat, zda chce chybu zadat manuálně, nebo automaticky.

  Automatické zadání umožňuje dvě možnosti a to buď zavedení opravitelných náhodných chyb, nebo zavedení zcela náhodné chyby, kterou dekodér nemusí být schopen opravit.

  Textové pole „Pozice zavedených chyb“ uživateli ukazuje, na jakých pozicích nastala chyba. Výpis těchto pozic je orientovaný, tak, jak s nimi pracuje Matlab. Tedy z leva doprava.

  Poslední textové pole „Bloky s chybou“ ukazuje výsledná slova, obsahující chybu a realizuje nám zprávu vstupující do dekodéru.

  Tato dvě poslední pole se vyplní po stisku tlačítka [Zakóduj] a toto nám opět zpřístupní tlačítka [Dekóduj] (Bod 7) a [Ulož].

  Tlačítko pro uložení uloží následující bloky: Zakódovaná zpráva, zpráva s chybou a pozice zavedených chyb.

- **Bod7 – Pole oblasti dekódování**

  Pole dekódování obsahuje 4 pro uživatele pouze informační textové bloky. První textový blok „Syndromy“ obsahuje vypočtené syndromy z přijaté zprávy.

  Druhé textové pole „Lokátory chyb“ obsahují vypočítané lokátory chyb pro jednotlivá kódová slova.

  Další textové pole „Pozice vzniklých chyb (výpočtem)“ ukazuje, na kterých pozicích v kódovém slově vznikla chyba. Toto pole má sloužit pro kontrolu správnosti dekódování, a pokud jsme dekódovali správně, mělo by mít stejný obsah jako pole „Pozice vzniklých chyb“ (Bod 6).

  Posledním textovým polem v této oblasti aplikace je pole „Opravené bloky zprávy“. V tomto poli je ukázáno, jak vypadají pouze informační části zakódované opravené přijaté zprávy. Opět by se při správném dekódování měli shodovat s polem „Převedeno do bloků“ (Bod 5).

  Všechna tato pole se vyplní při stisku tlačítka [Dekóduj] a toto nám zpřístupní poslední tlačítko [Ulož].

  Poslední tlačítko pro uložení slouží k uložení následujících prvků: Vstupní text, vypočítané syndromy, vypočítané lokátory chyb, vypočítané pozice chyb, opravené bloky zprávy a dekódovaná zpráva.

- **Bod8 – Pole dekódované zprávy**
  V tomto posledním textovém bloku, který je vyplněn také na stisk tlačítka [Dekóduj] je vypsána dekódovaná zpráva, která by při správném dekódování se měla shodovat se zprávou na výstupu.
- **Bod9 – Pole menu možnosti**
  V této oblasti se nachází rozbalovací menu s položkami import a export. Položka import nám slouží k načtení rozsáhlejšího textu pomocí výběru textového souboru. Tento text bude po výběru souboru zobrazen do textového pole pro zadání vstupního textu. Položka export slouží k uložení všech údajů, které při kódování/dekódování uživatel spustil.

5.1 Funkce aplikace

V Tato aplikace využívá kromě interních funkcí Matlabu následující vytvořené funkce.

- **okno.m**
  Toto je hlavní funkce programu. Obsahuje všechny funkce GUI, které jsou volány jednotlivými prvky aplikace a také jsou zde umístěny vytvářecí funkce všech prvků, které aplikace obsahuje.
- **lokatorychyb.m**
  Toto je funkce, která je volána během dekódování poté, co jsou známy syndromy. Do funkce vstupují právě tyto syndromy, minimální vzdálenost kódu a řád Galoisova tělesa. Po dokončení výpočtu tato funkce sestaví textový výstup těchto lokátorů chyb, aby mohl být vložen do příslušeného textového pole. Výstupem jsou dvě matice. Matice lokátorů chyb pro výpočet a matice ve formátu string.
- **minimaly.m**
  Tato funkce podle zadaného parametru *m*, vrátí matici minimálních polynomů.
  Do funkce tedy vstupuje pouze řád GF a výstupem je matice minimálních polynomů.
- **opravychyby.m**
  Funkce, která opraví matici zakódované zprávy s chybou pomocí zjištěné matice vektorů chyb.
  Vstupy funkce jsou matice chybových vektorů, matice zprávy s chybou a parametr kódu *n*. Výstupem funkce je matice zprávy s opravenými chybami.
- **prevoddoascii.m**
  Prevoddoascii je funkce, která převádí textovou podobu zprávy do binární podoby a následně do podoby polynomu. Nakonec sestaví textovou podobu, kde každému znaku přiřadí jeho binární podobu. Toto je poté použito pro výpis do textového pole „Zpráva v ascii“.
  Do funkce vstupuje pouze vstupní text pro kodér. Výstupem je matice binární podoby znaků, matice polynomiální podoby znaků a matice textu do textového pole.

- **prevodgenpoly2rovnice.m**
  Tato funkce slouží k převedení vektoru generujícího polynomu do textové podoby s proměnou *x*. Jedná se o funkci sloužící pouze k estetické úpravě.
  Do funkce vstupuje vektor generujícího polynomu a výstupem je textový vektor generujícího polynomu.
- **prevodgf2num.m**
  Jednoduchá funkce, která převede proměnnou GF do proměnné numerického charakteru. S proměnnou GF nejsou některé ostatní funkce schopny pracovat,
  Vstupem funkce je sloupcový vektor proměnné GF a řád tohoto GF. Výstupem pak sloupcový vektor převedený do num.
- **prevodstr2polynom.m**
  Funkce, která převede vektor v podobě textu do formy polynomu.
  Na vstupu této funkce musí být pouze text obsahující 0 nebo 1 bez mezer. Druhým vstupem je délka tohoto polynomu. Výstupem je vektor požadované délky.
- **prevodzblokudozpravy.m**
  Tato funkce je volána u konce dekódování. Slouží k převedení opravených informačních bloků zprávy k sestavení ascii slov, abychom je mohli převést na text.
  Vstupem funkce je matice informačních slov opravené zprávy a výstupem je matice slov v ascii podobě.
- **vypocetvektoruchyb.m**
  Tato funkce nám slouží k vypočtení chybového vektoru z předem vypočítaných lokátorů chyb. Princip výpočtu je popsán v kapitole 4. Tato funkce také opět vytvoří textovou podobu těchto chybových vektorů.
  Vstupem funkce je matice lokátorů chyb a opět řád GF. Výstupem pak matice chybových vektorů a textová matice sestavených chybových vektorů.
- **vypoctenisyndromu.m**
  Pomocí této funkce se počítají syndromy přijaté zprávy. Tato funkce opět sestaví textovou podobu těchto syndromů pro vložení do textového pole „Syndromy“.
  Do funkce vstupují vzestupně seřazené vektory exponentů kořenů minimálních polynomů. Dále parametr kódu *n*, a matice kódových slov se zavedenou chybou. Výstupem jsou pak dvě matice, kde první obsahuje vypočtené syndromy a druhá sestavený text do pole „Syndromy“.
- **vytvorgenpoly.m**
  Tato funkce nám ze zadaných parametrů kódu *k*, *n* sestaví generující polynom a zjistí Hammingovu vzdálenost. Dále také zjistí korekční schopnost kódu a sestaví seřazené exponenty kořenů použitých minimálních polynomů.
  Vstupem funkce jsou parametry *k* a *n*. Výstupem je generující polynom, seřazené exponenty kořenů, kódová vzdálenost a korekční schopnost kódu.

- **zakoduj.m**
  Tuto funkci program volá v cyklu, kdy je nutné zakódovat jednotlivé vektory informačních slov.
  Do funkce vstupují parametry kódu *n* a *k*. Dále pak informační slovo ve formě řádkového vektoru a generující polynom. Výstupem je potom vektor zakódovaného slova.
- **zavedenichybydozpravy.m**
  Toto je jednoduchá funkce, která sečte dvě matice. Matici chybových vektorů a matici zakódované zprávy. Následně vytvoří textovou podobu zprávy s chybou.
  Vstupem funkce jsou matice chybových vektorů a matice zakódované zprávy. Výstupem je matice zprávy s chybou a textová podoba této matice.
- **zavedenidobloku.m**
  Jednoduchá funkce, která zprávu z ascii převede do bloků o požadované délce podle parametru *k*.
  Vstupem funkce je tedy matice polynomů binární hodnoty ascii a parametr kódu *k*. Výstupem je matice zprávy převedené do bloků.
- **zavedeninahodnechyby.m**
  Tato funkce je použita pro automatické zavedení chyby. Uživatel pouze zvolí, zda chce, aby se jednalo o chybu náhodou nebo náhodou opravitelnou. Poté je vygenerován počet chyb právě podle parametru, zda musí být opravitelná. Dále se vygeneruje vektor pozic chyb a ten se seřadí. Poté budou vyloučeny opakující se prvky. Nakonec se vytvoří dvě matice chybových vektorů, textová a numerická .
  Vstupem funkce jsou parametr *k*, korekční schopnost kódu, počet kódových slov a informace o opravitelnosti chyb. Výstupem jsou pak dvě matice. První je numerická obsahující chybové vektory, druhá obsahuje textovou podobu chybových vektorů.
- **zjistenivektoruzavedenechyby.m**
  Toto je funkce volaná funkcí zavedenichybydozpravy.m. Tato funkce porovná dvě matice, kde první je tvořena kódovými slovy bez chyby a druhá kódovými slovy s chybami. Z tohoto porovnání následně sestaví chybový vektor. Nakonec opět vytvoří textovou podobu do textového pole.
  Vstupem funkce jsou matice kódových slov bez chyb a matice kódových slov s chybou. Výstupem potom jsou textová matice chybového vektoru a numerická matice chybového vektoru.

Závěr

Tato práce je zaměřena na skupinu bezpečnostních kódů BCH. Tato zkratka vychází ze objevitelů těchto kódů. Byli to Hocquengh v roce 1959, a v roce 1960 Bosem a Ray-Chaudhurim nezávisle na Hocquenghovi. Princip všech kódů z oblasti bezpečnostních kódů spočívá v rozšíření přenášené informace o několik kontrolních bitů, díky kterým jsme schopni na příjímací straně dekódovat zprávu i v případě, že se vyskytla chyba při přenosu. BCH kódy jsou velmi rozšířené v oblasti výpočetní techniky. Toto způsobuje právě jejich důležitá vlastnost, že lze velmi variabilně volit parametry tohoto kódu. Tyto parametry jsou např. délka kódového nebo informačního slova, počet opravitelných chyb a další.

V této práci byly nejprve popsány matematické podklady pro kódování a dekódování z obecného pohledu a také konkrétně pro BCH kódy. Detailně zde byl rozebrán postup kódování i dekódování. Tento teoretický postup byl následně dokázán praktickou ukázkou výpočtu.

Aplikace kodéru/dekodéru, která byla předmětem této bakalářské práce, odpovídá zadání. Obsahuje vizuální výstup nejdůležitějších kroků kódování i dekódování. Je navržena pouze pro kódování textového vstupu bez diakritiky. Dále je v této aplikaci možnost načítat textová data pro kodér z externího textového souboru. Opět je nutné, aby takto vložený text neobsahoval znaky s diakritikou. Uživatel této aplikace si může všechny získané výsledky ukládat do textových souborů jako celek, ale je možnost uložení i jednotlivých kroků. Toto může uživateli sloužit pro pozdější porovnávání výsledků. Kód aplikace je ošetřen proti většině chyb, které by uživatel mohl zadat.

Tato aplikace je tvořena souborem funkcí, které jsou popsány v kapitole 5.1 a jsou umístěny na přiloženém CD. Většina těchto funkcí je použitelná i pro jiné aplikace, pouze se uživatel musí seznámit se vstupy a výstupy těchto funkcí.

Jedna z nevýhod aplikace je její závislost na instalaci Matlabu v PC. Toto by se dalo odstranit použitím kompilátoru. Nejlépe zkompilovat do programovacího jazyka např. Java.

Aplikace by mohla být rozšířena o možnost vkládání textu s diakritikou a zadruhé o možnost kódování i jiných vstupů, než pouze textu. Další z možností doplnění této aplikace by mohla být grafická ukázka schéma kodéru a dekodéru podle zvoleného kódu.

Aplikace je navržena jako výuková pomůcka pro předmět teorie informace. Umožňuje uživateli názornou ukázku všech kódovacích / dekódovacích kroků či ověření získaných výsledků.

Literatura

Adámek, RNDr. Jiří. 1989. *Kódování.* Praha : SNTL, 1989.

Costello, Daniel J. a Lin, Shu. 1983. *Error Control Coding: Fundamentals and.* místo neznámé : Prentice-Hall, 1983. ISBN: 0-13-042672-5.

Číka, Petr. 2006. Protichybové zabezpečení BCH kódem. *Elektrorevue.* [Online] 13. 3 2006. [Citace: 7. 3 2013.] http://www.elektrorevue.cz/clanky/06015/index.html. ISSN 1213-1539.

Hoffman, D. G., a další. 1992. *Coding theory the essentials.* New York : Marcel Dekker, Inc., 1992. ISBN 0-8247-8611-4.

Moon, T. K. 2005. *Error Correction Coding: Mathematical Methods and Algorithms.* místo neznámé : John Wiley & Sons, 2005. ISBN: 0-471-64800-0.

Moreira, Jorge Castiñeira a Farell, Patrick Guy. 2006. *Essentials of error-control coding.* Chichester : John Wiley & Sons, 2006. ISBN-13 978-0-470-02920-6.

Němec, Karel. 2005. Protichybové kódové zabezpečení s Bose-Chaudhury-Hocquenhemovými kódy. *Elektrorevue.* [Online] 14. 3 2005. [Citace: 10. 3 2013.] http://www.elektrorevue.cz/clanky/05015/index.html. ISSN 1213-1539.

Peterson, Wesley W. a Weldon, E. J. 1972. *Error-Correcting Codes.* [e-kniha] Cambridge : MIT Press, 1972. 0-262-16-039-0.

The MathWorks, Inc. 1994. *MathWorks.* [Online] The MathWorks, Inc., 1994. http://www.mathworks.com/.

Zaplatílek, Karel a Doňar, Bohuslav. 2003. *MATLAB pro začátečníky.* Praha : BEN - technická literatura, 2003. ISBN 80-7300-095-4.

Příloha A – Galoisovo těleso GF(16)

Galoisovo těleso GF(24)			
exponent	**zbytek po dělení**	**binární reprezentace**	**reprezentace v matlabu**
α^0	α^0	0001	1
α^1	α^1	0010	2
α^2	α^2	0100	4
α^3	α^3	1000	8
α^4	$\alpha+1$	0011	3
α^5	$\alpha^2+\alpha$	0110	6
α^6	$\alpha^3+\alpha^2$	1100	12
α^7	$\alpha^3+\alpha+1$	1011	11
α^8	α^2+1	0101	5
α^9	$\alpha^3+\alpha$	1010	10
α^{10}	$\alpha^2+\alpha+1$	0111	7
α^{11}	$\alpha^3+\alpha^2+\alpha$	1110	14
α^{12}	$\alpha^3+\alpha^2+\alpha^2+1$	1111	15
α^{13}	$\alpha^3+\alpha^2+1$	1101	13
α^{14}	α^3+1	1001	9
α^{15}	α^0	0001	1

Příloha B – Zdrojový kód funkce vypoctenisyndromu.m

```
function [syndromy,syndromystr]=vypoctenisyndromu (serazenesyndromy,n,vstupnimatice)
%syndromy
syndromystr='';
m=log2(n+1);
velikostmatice=size(vstupnimatice);
syndrom=zeros(1,m);
for (radek =1:velikostmatice(1,1))
for (poradi =min(serazenesyndromy):max(serazenesyndromy)) %do sloupce syndromy
syndrom(poradi,:)=zeros(1,1:m);
for(i = 0:n-1)
   if (vstupnimatice(radek,i+1)==1)  %pokud je prijate slovo bit po bitu nastaven na "1"
         %tak přičtu postupně hodnoty z GFtuple
   syndrom(poradi,:)=mod((syndrom(poradi,:)+gftuple(i*poradi,m)),2);
   end;
end;
   exponentsyndromu(poradi,1)=-Inf;
   for (j = 0:14)
      gftuple(j,m)
      if (gftuple(j,m)==syndrom(poradi,:))
         exponentsyndromu(poradi,1)=j;   %vypočte exponenty syndromu
         break;
      end;
   end;
   pomocne2=bin2dec(fliplr(num2str(syndrom(poradi,:))));    %převod z polynomu na binár a do dekadické
hodnoty-> binár se musí otočit aby odpovídalo syntaxi
   vypoctenesyndromy(poradi,1)=pomocne2;    %binární hodnota syndromů dekadicky vyjadřená
end;
syndromy(radek,:)=vypoctenesyndromy'%uloží binární reprezentaci zbytku po dělení (syndrom)
exponenty=exponentsyndromu';
pomocny=['w' num2str(radek) ':'];
for (delka=1:length(exponenty))
   if (exponenty(delka)==-Inf)
      if (delka==length(exponenty))
         pomocny=strcat(pomocny,[' S' num2str(delka) '=0;']);
      else
         pomocny=strcat(pomocny,[' S' num2str(delka) '=0,']);
      end;
   else
      if (delka==length(exponenty))
         pomocny=strcat(pomocny,[' S' num2str(delka) '=a^' num2str(exponenty(delka)) ';']);
      else
         pomocny=strcat(pomocny,[' S' num2str(delka) '=a^' num2str(exponenty(delka)) ',']);
      end;
   end;
end;
syndromystr=strvcat(syndromystr,pomocny);
end;
```

Příloha C – Zdrojový kód funkce lokatorchyb.m

Pozn. Zdrojový kód obsahuje pouze část výpočtu, bez převedený výsledku do tvaru pro textovou buňku

```
function [lokatory,lokatorystr]=lokatorchyb(syndromy,dmin,m)

velikostmatice=size(syndromy)
opravychyb=(dmin-1)/2

for (slovo=1:velikostmatice(1,1))

for (i=1:opravychyb)
for (j=1:opravychyb)
soustava(i,j)=syndromy(slovo,j+i-1)
end;
pravysloupec(i,1)=syndromy(slovo,opravychyb+i)
end;
soustava=gf(soustava,m)
pravysloupec=gf(pravysloupec,m)

%odstranění lineárně závislých rovnic
[radky,sloupce]=size(soustava)
hodnost = rank (soustava)
if (opravychyb~=rank(soustava))
   soustava=soustava([1:hodnost],[1+(sloupce-hodnost):sloupce])
   pravysloupec=pravysloupec([1:hodnost],1)
end;
det(soustava)

if (rank (soustava)==0)
   Y=0
elseif (det(soustava)==0)
   soustava=soustava(1,:)
   pravysloupec=pravysloupec(1,1)
   X=soustava\pravysloupec
   Y=prevodgf2num(X,m)
else
   X=soustava\pravysloupec
   Y=prevodgf2num(X,m)
end;

for(s=1:length(Y))
lokatory(slovo,s)=Y(s,1)
end;
end;
```