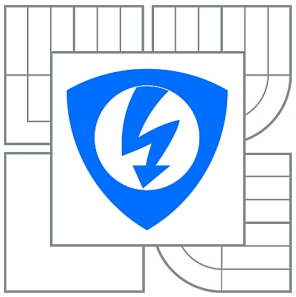
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

PROTICHYBOVÉ SYSTÉMY S PROKLÁDÁNÍM

ANTIERROR SYSTEMS WITH INTERLEAVING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

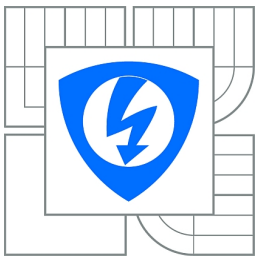
AUTOR PRÁCE — Bc. JAKUB PACHER

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAREL NĚMEC, CSc.

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Jakub Pacher	***ID:***	78392
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Protichybové systémy s prokládáním

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte protichybový systém s bitovým prokládáním, který zabezpečí přenos dat proti shlukům chyb b je menší nebo rovno 64 bitů, při bezchybném intervalu mezi shluky chyb A je větší nebo rovno 640 bitů, při informační rychlosti použitého kódu R větší než 0,5. Pro tento systém vypracujte návrh realizace tak, aby byl schopen pracovat jako individuální systém. Ověřte funkční schopnost tohoto systému metodou, kterou považujete pro tento návrh za nejvhodnější.

DOPORUČENÁ LITERATURA:

[1] HEEGARD,CH.-WICKER,S.B. Turbocoding. Kluwer Academic Publisher. Boston, Dortrecht, London, 1999, ISBN 0-7923-8378-8.
[2] VLČEK,K. Komprese a kódová zabezpečení v multimediálních komunikacích. BEN - Technická literatura, Praha 2004, ISBN 80-7300-134-9.
[3] HOUGHTON,A. Error Coding for Engineers. Kluwer academic Publishers. Boston, Dordrecht, London. 2001.

Termín zadání:	29.1.2010	***Termín odevzdání:***	26.5.2010

Vedoucí práce: doc. Ing. Karel Němec, CSc.

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

UPOZORNĚNÍ:

ABSTRAKT

Abstrakt práce v češtině

Tato práce se zabývá protichybovými systémy s prokládáním. Nejdříve je podán přehled o často používaných zabezpečovacích kódech. Dále jsou popsány základní dvě techniky prokládání spolu s jejich porovnáním. V další části textu již dochází k rozboru kódů vyhovujících zadání. Je vybrán optimální systém, jehož funkčnost je nejprve ověřena v simulačním prostředí MATLAB a následně je navržena funkční aplikace v jazyce C++ pro přenos zabezpečených BMP obrázků.

KLÍČOVÁ SLOVA

Klíčová slova v češtině

samoopravný kód, prokládání, shluk chyb, protichybový kódový systém, Fireův kód, Hammingův kód, BCH kód, BMP obrázek

ABSTRACT

Abstract in English

This work involves in anti-error coding systems with interleaving. At first is given summary of high-frequency use error correction codes. Below there are described two basic techniques of interleaving and their confrontation. The next text is focusing on survey and characteristics of codes which conform to submission. After selection of optimal system is verified its function in MATLAB environment. Final step is creation of functional application in C++ environment. This application serves to transmission of error correction BMP pictures.

KEYWORDS

Keywords in English

error-correcting code, interleaving, burst error, anti-error coding system, Fire code, Hamming code, BCH code, BMP picture

PACHER J. *Protichybové systémy s prokládáním*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 46 s. Vedoucí diplomové práce doc. Ing. Karel Němec, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Protichybové systémy s prokládáním“ 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 tohoto projektu 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.

V Brně dne ……. ……..

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval svému vedoucímu diplomové práce doc. Ing. Karlu Němcovi, CSc. za příkladnou metodickou, pedagogickou a odbornou pomoc a za další cenné rady při zpracování mé diplomové práce.

V Brně dne ……. ……..

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 ÚVOD

V současné době prudkého rozmachu informačních technologií se přenos dat pomocí různých systému stal každodenní potřebou většiny z nás. Ať už se jedná o prohlížení e-mailu či komunikaci pomocí mobilního telefonu, vždy je hlavním cílem zajistit, aby z přenesených dat bylo možno na přijímací straně sestavit původně vyslanou zprávu. Při přenosu pomocí kabeláže, či radiovými vlnami nebo optickými vlnami však dochází k různým rušivým vlivům, které zavádí chyby a užitečný signál znehodnocují.

Touto problematikou se zabývá teorie kódování, která vznikla již ve čtyřicátých letech pracemi Shannona [1], věnovanými teorii informace, a Hamminga s Golayem, kteří konstruovali první lineární kódy.

Základním principem zabezpečení informace vůči chybám je přidání jisté míry nadbytečnosti, pomocí které je pak možno na straně přijímače odvodit původně odeslanou zprávu. Matematický aparát, který toto zajišťuje, se nazývá zabezpečovací kód a je definován operacemi, které se provádí se vstupními daty pro odvození nadbytečné (zabezpečovací) posloupnosti.

Jako doplněk k zabezpečovacímu kódování je pro zvýšení odolnosti protichybového kódového systému PKS zejména vůči shluku chyb možno s výhodou využít metody prokládání dat. Tato zajistí změnu polohy prvků zprávy tak, že se následně změní distribuce chyb ze shlukových chyb na chyby samostatné, které lze účinněji odstranit či potlačit.

1.1 Cíl práce

Cílem práce je navrhnout protichybový kódový systém, který zabezpečuje přenos dat proti shlukům chyb za použití vhodného zabezpečovacího kódů v kombinaci s technikou bitového prokládání. Nejdříve se seznámíme s možnými druhy chyb a kritérii pro výběr vhodného zabezpečovacího kódu. Dále detailně popíšeme možnosti realizace bitového a okrajově i konvolučního prokládání. V závěru práce se budeme zabývat konkrétním zadáním. Při výběru vhodného kódu bude brán zřetel nejen na zabezpečovací schopnosti tohoto kódu, ale také na složitost procesu kódovaní a dekódování.

Pro ověření a následnou realizaci se nabízí více možností, proto si je nejdříve rozebereme a zvolíme tu, která bude z časové a finanční stránky nejméně náročná. Jedná se tedy o výběr a rozbor vhodného kódu a vhodné techniky prokládání spolu s ověřením funkčnosti a následnou realizací navrženého systému.

2 ZABEZPEČENÍ SAMOOPRAVNÝM KÓDEM

Díky rušivým vlivům při přenosu číslicového signálu vznikají chyby. Pro příjemce má však význam pouze bezchybně přijatá zpráva a tak je třeba přenos nějakým způsobem zabezpečit. Využít při tom můžeme samoopravných kódů, které vlivem zavedené redundance dokáží na přijímací straně chyby odstranit.

Při opravě chyb způsobem FEC (forward error correction) dochází v dekodéru k přiřazování přijaté posloupnosti symbolů nejpravděpodobnější posloupnosti, která byla vysílačem vyslána.

Další princip opravy vychází z žádosti o opětovné vyslání přijaté posloupnosti ARQ (automatic repeat request). Tato žádost se vysílá po zpětném kanálu, který je vytvořen mezi přijímačem a vysílačem. Jelikož ze zadání nevyplývá, že je tento kanál k dispozici, nebude tato varianta dále brána v úvahu.

2.1 Základní druhy chyb

Chyba v datovém kanále vzniká díky zmenšení odstupu výkonu přenášeného signálu od výkonu šumu. Poté, díky překročení rozhodovací úrovně, je na přijímací straně v některém vzorkovacím okamžiku chybně vyhodnocen přijímaný signálový prvek.

Nezávislé chyby – Jsou chyby, které jsou relativně rovnoměrně rozloženy v přenášené zprávě. V daném úseku se může nalézat jedna chyba, nazýváme ji jednoduchou, anebo více chyb, nazývaná vícenásobná. Poté veličiny, které tyto chyby popisují jsou průměrný počet chybně přenesených prvků v celkovém počtu prvků, průměrný počet bezchybných prvků nacházejících se mezi dvěma chybnými prvky, atd.

Shluky chyb – jsou takové části zprávy, kde hustota chybně přenesených prvků je mnohem větší než ve zbytku zprávy. Pro shluky jsou definovány veličiny jako průměrná délka shluku, průměrná vzdálenost mezi shluky, průměrný počet chyb ve shluku, atd.

Hlavní otázka je, zda při objevení se na kanálu chyb nezávislých i shluku chyb, zařadit nezávislou chybu do shluku, či ne. Praxe ukazuje, že realizace zdánlivě složitějšího protichybového kódu zabezpečujícího proti dlouhým shlukům chyb je výhodnější, než použití kódu pro zabezpečení proti nezávislým chybám, tím pádem se nezávislé chyby přiřazují ke shluku [2].

2.2 Princip zabezpečovacího kódování

V datovém kanále nám proudí množina signálových prvků. Podle [2] si tuto množinu rozdělíme na posloupnosti jisté délky. Příjemce musí řešit otázku, zda přijatá posloupnost byla přenesena správně či ne. Toho docílíme tak, že všechny možné kombinace posloupností rozdělíme na užívané, přenáší samotné informace, a neužívané, díky nimž zjistíme, zda přijatá posloupnost byla přenesena chybně. Poté tedy, při přijetí neočekávané posloupnosti, určí příjemce za očekávanou, neboli správnou, posloupnost tu, která se od přijaté posloupnosti liší v nejmenším počtu prvků.

Rozdělení na užívané a neužívané posloupnosti se řeší na základě tzv. Hammingovy vzdálenosti. Hammingova vzdálenost d určuje počet prvků ve kterých se dvě posloupnosti stejné délky liší. Tudíž při zvyšování Hammingovy vzdálenosti vznikne tak velký počet možných značek, že je budeme moci rozdělit na užívané a neužívané. Tímto je zajištěna odolnost vůči chybám a na tomto je založen princip zabezpečení přenášené posloupnosti kódem.

Zabezpečovací kódování je tedy založeno na zvyšování minimální Hammingovy vzdálenosti d_{min}, kdy k prvkům nezabezpečené zprávy přidáváme prvky zabezpečovací a to podle pravidel definujících daný zabezpečovací kód. Toto se využívá u systematických blokových kódů.

Jiný způsob je založen na výběru takových kódových kombinací z celkového počtu kombinací, které splňují podmínku d_{min}. Tím opět získáváme množinu používaných a nepoužívaných kódových kombinací. Použití u nesystematických blokových kódů.

Podle velikosti minimální Hammingovy vzdálenosti můžeme rozlišit kódy, které chybu vůbec neodhalí, chybu detekují či chybu korigují.

Detekce chyby - změna na jednom místě kombinaci prvků je detekována, jelikož užívaná kombinace přejde v neužívanou, t_2 značí počet detekovaných chyb.

$$d_{min} = t_2 + 1, \tag{2.1}$$

Korekce chyby - změna na jednom místě je detekována, dále však může dojít i ke korekci, kdy se najde taková kódová kombinace, která má od chybné nejmenší Hammingovu vzdálenost, t_1 značí počet korigovaných chyb.

$$d_{min} = 2t_1 + 1, \tag{2.2}$$

Smíšená schopnost – spojuje dohromady předchozí dvě zmíněné možnosti.

$$d_{min} = t_1 + t_2 + 1. \quad (2.3)$$

2.3 Kritéria pro výběr vhodného kódu

Při návrhu PKS chceme docílit přizpůsobení kódu na kanál, tzn. navrhnout takový korekční kód, který je schopen opravit chyby vznikající při přenosu. Vychází se přitom z distribuce chyb ve sdělovacím systému, kterou můžeme získat např. v podobě pravděpodobnostním charakteristik chybovosti daného kanálu.

Při výběru optimálního korekčního kódu musíme dále brát v úvahu i konkrétní vlastnosti a parametry daného kódu. Jsou jimi informační poměr kódu *R*, celková složitost kodeku *S* a celkové zpoždění *Z*.

Informační poměr kódu *R* – je dán poměrem délky vstupní posloupnosti *k* a délky výstupní posloupnosti *n*. Je zřejmé, že čím více se tento poměr blíží k 1, tím méně nadbytečnosti daný kód zavádí a tím způsobuje menší snížení přenosové rychlosti.

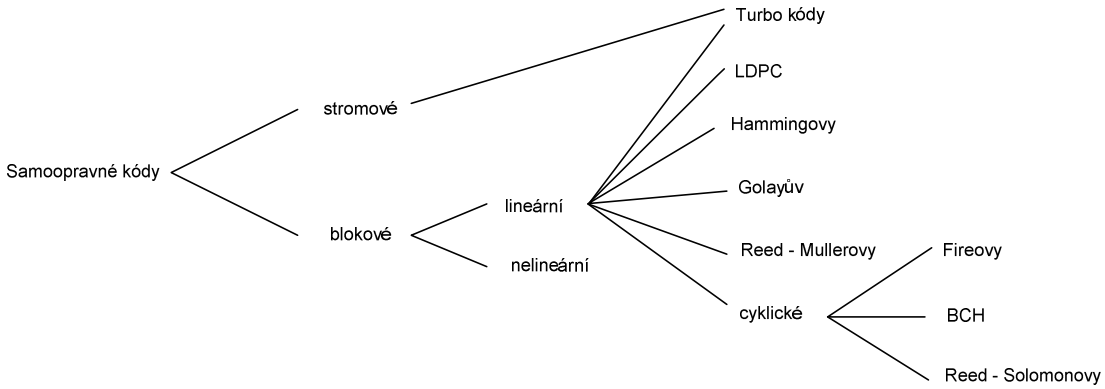
$$R = \frac{k}{n}, \quad (2.4)$$

Celková složitost kodeku *S* – složitostí se rozumí počet paměťových buněk PB, které jsou v daném PKS použity.

Celkové zpoždění *Z* – úzce souvisí s celkovou složitostí kodeku. Můžeme jej vyjádřit počtem impulsů časové základny, které nastanou mezi vstupem a výstupem určitého informačního bitu přenášené zprávy v PKS.

2.4 Přehled některých zabezpečovacích kódů

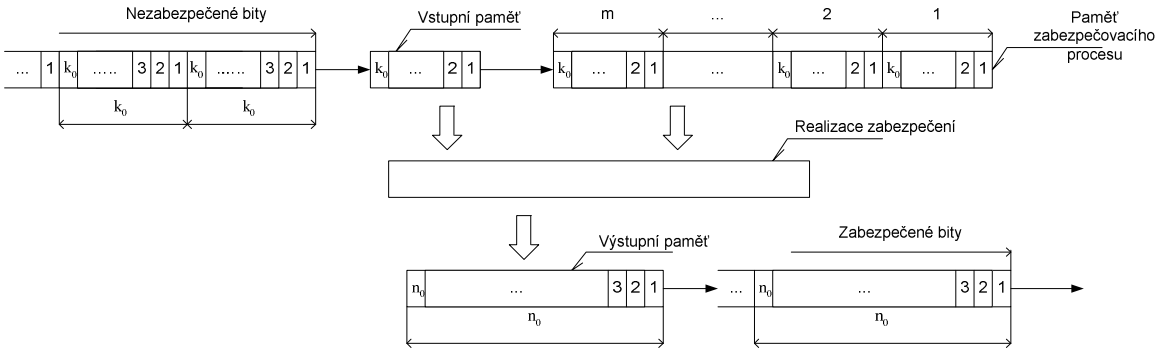
Pro přehlednost zde budou uvedeny některé zabezpečovací kódy, jejich skupiny a jejich základní vlastnosti. Samoopravné kódy můžeme rozdělit do dvou velkých skupin, a to blokové a konvoluční. Další dělení je naznačeno na *obr. 2.1*.



Obr. 2.1: *Rozdělení některých samoopravných kódů*

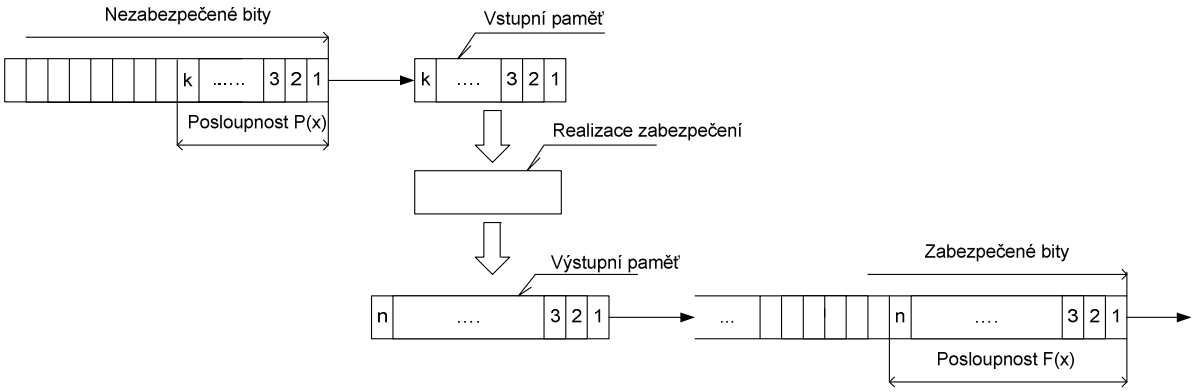
Pro přesnější orientaci v *obr. 2.1* uvádíme stručnější popis těchto kódů vycházející z [2].

Stromové kódy - Název vychází ze způsobu vyjádření kódovacího procesu, kdy se nejčastěji používá graf typu strom. Hlavní rozdíl oproti blokovým kódům spočívá v realizátoru, který navíc obsahuje paměť zabezpečovacího procesu. Do ní se ukládají úseky nezabezpečené zprávy k_0 z m předcházejících časových okamžiků. Pro samotnou realizaci zabezpečení se poté nepoužívají pouze prvky ze současného k_0, ale právě i prvky, které jsou uloženy v paměti zabezpečovacího procesu. Na výstupu získáváme úseky zabezpečené zprávy n_0.



Obr. 2.2: *Realizace zabezpečení stromovým kódem*

Blokové kódy - Jak již sám název napovídá, pracují s tzv. bloky. Jsou to úseky vyňaté z celkové posloupnosti signálových prvků. Úseky nazýváme kódové kombinace.
Na vstup realizátoru kódu přichází mnohočlen nezabezpečeé zprávy *P(x)* složen z *k* prvků nezabezpečeného toku dat. Ten je přeměněn na mnohočlen zabezpečené zprávy *F(x)*, kdy se ke *k* prvkům pomocí jistého algoritmu přidá *r* prvků zabezpečovacích. Na přijímací straně poté do dekódovacího zařízení vstupuje mnohočlen přenesené zprávy *J(x).*



Obr. 2.3: *Realizace zabezpečení blokovým kódem*

Hammingovy kódy - Jedná se o blokové korekční kódy, dokáží opravit jednu chybu a jejich minimální Hammingova vzdálenost je $d_{min} = 3$. V případě Hammingových kódů pro smíšené zabezpečování, kdy se přidá celková kontrola parity (tj. přidá se jeden kontrolní symbol navíc), dochází ke korekci dvou chyb a $d_{min} = 4$. Jejich hlavní výhodou je, že dokáží opravit jednoduché chyby s co nejmenší myslitelnou redundancí. Takovéto kódy se nazývají kódy perfektní.

Golayův kód G_{23} – Kód binární lineární (23,12) s dvanácti informačními a jedenácti kontrolními znaky. Jedná se o kód perfektní pro trojnásobné chyby. Může být rozšířený na kód (24,12) přidáním celkové kontroly parity. Používá se v aplikacích, kde není zapotřebí velká průchodnost.

Fireovy kódy – Cyklický kód opravující shluky chyb určený vytvářecím mnohočlenem tvaru $G(x) = N(x) \cdot Q(x)$, kde $N(x)$ je nerozložitelný mnohočlen řádu m. Mnohočlen $N(x)$ je řádu m dělí-li mnohočlen $x^m + 1$ a žádný jiný mnohočlen nižšího stupně. Mnohočlen $Q(x)$ je tvaru ve $x^c + 1$.

Bose-Chaudhuriho-Hocquenghemovy kódy – Cyklické blokové binární kódy, které zvládají opravu vícenásobných chyb. Mezi jejich přednosti patří velká volitelnost parametrů, dobrý vztah mezi počtem informačních znaků a počtem opravovaných chyb a detailně vypracované dekódovací metody. Jejich název je zkracován na BCH kódy.

Reed-Solomonovy kódy – Vychází z BCH kódů a jejich předností je, že mají nejlepší opravné schopnosti pro kód dané délky. Ve srovnání s BCH kódy mají však složitější způsob dekódování.

LDPC kódy – Blokové kódy, které mají dlouhé kódové kombinace a jsou charakteristické řídkou kontrolní maticí. Objeveny byly již v roce 1961, ale pak byly na dlouho dobu

zapomenuty díky složitému způsobu dekódování. V současné době dekódování probíhá např. pomocí iterativní metody. Představují alternativu k turbo kódům.

Turbo kódy – Tyto kódy vznikají paralelním zřetězením dvou nebo více kódů za použití prokládání. Mohou být při tom využiti jak blokové tak i konvoluční kodéry. Jejich objevení se datuje rokem 1993 a s jejich využitím se lze přiblížit Shannonově mezi.

3 SYSTÉMY S PROKLÁDÁNÍM

Proces prokládání je standardní způsob zpracování signálu, který se používá v různých komunikačních systémech. Prokladač je zařízení, které vstupní symboly jistým způsobem přehází a tím na výstupu získáváme stejné symboly avšak ve změněném pořadí. Klasické užití prokládacích metod je zajištění „náhodného“ rozprostření pozic chyb. Pokud se tedy během přenosu objeví shluky chyb, díky němuž by mohla být překročena zabezpečovací schopnost kódu, rozprostře se tento shluk do chyb, které je již možné opravit obvyklými korekčními kódy. Z principu metody však vyplývá, že se do přenosového procesu zavádí dopravní zpoždění. Toto má za následek snížení přenosové rychlosti. Blokové schéma obecného systému s prokládáním je zobrazeno na *obr. 3.1*.

Obr. 3.1: *Blokové schéma obecného systému s prokládáním*

3.1 Bitové prokládání

Princip bitového prokládání je podle [3] následující. Tok nezabezpečených bitů vstupuje do kodéru, kde se k němu přidá (n - k) zabezpečovacích prvků pomocí nichž lze v dekodéru opravit až t chyb. Následuje ukládání kódových kombinací po řádcích do prokládací matice, která má rozměr n x j. Pro parametr j, tedy počet řádků matice, musí platit

$$j \geq \frac{b}{t}, \tag{3.1}$$

kde b značí velikost shluku chyb, které chceme opravit a t je počet chyb, které je náš kód schopen opravit.
Pro parametr n, tedy počet sloupců matice musí platit

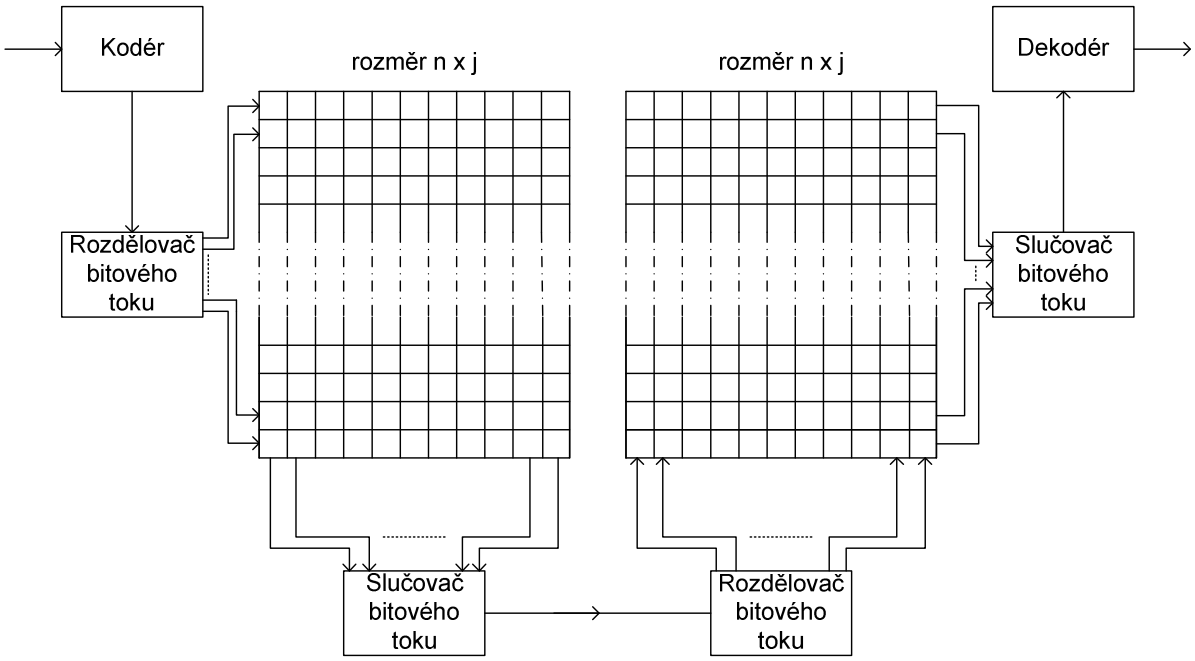
$$n \leq \frac{A+b}{j}, \tag{3.2}$$

kde A značí minimální délku bezchybného intervalu mezi shluky chyb.

Z matice jsou poté prvky čteny po sloupcích a přenášeny k přijímači. Zde se opět nachází prokládací matice, do které je zapisováno po sloupcích a čteno po řádcích. Vlivem prokladu je jakýkoliv shluk chyb délky *b* na přijímací straně rozložen na samostatné, či více násobné chyby, které už však dekodér dokáže opravit.

Z popisu vyplývá, že zpoždění vznikající prokladem bude rovno dvojnásobku rozměru prokládací matice, tedy

$$D_t = 2 \,.\, j \,.\, n \,. \qquad (3.3)$$



Obr. 3.2: *Schématické znázornění protichybového systému s bitovým prokládáním*

Možnou variantou blokového prokládání je tzv. diagonální prokládávání používané např. v systému GSM [8].

3.2 Konvoluční prokládání

Podle [4] se jedná o metodu, která je efektivnější oproti prokládání bitovému. Princip metody je znázorněn na *obr. 3.4*. Na vstup prokladače přicházejí jednotlivé bity, či několikabitové kódové symboly. Tyto se postupně přivádějí přepínačem do jednotlivých paměťových větví, přičemž určitý *i-tý* registr má o *m* symbolů větší paměť než registr předchozí. Po naplnění poslední větve se přepínač P1 vrací na začátek, čili dochází opět k přímému spojení mezi vstupem a výstupem prokladače, a celý cyklus se opakuje.

Na přijímací straně jsou symboly přiváděny přepínačem P3 opět k paměťovým větvím. Zde však každý *i-tý* registr má o *m* symbolů menší paměť, než registr předchozí. Díky tomu je celková doba zpoždění každého symbolu konstantní, a na výstupu inverzního prokladače se vyskytují symboly ve stejné sekvenci jako byly na vstupu prokladače. Důležité je, aby všechny přepínače pracovali synchronně.
Prokladač s hloubkou prokládání *D* má l_x větví. Pro jejich počet musí platit

$$l_x \geq \frac{b}{t}. \tag{3.4}$$

Pro velikost kódového slova musí opět platit podmínka

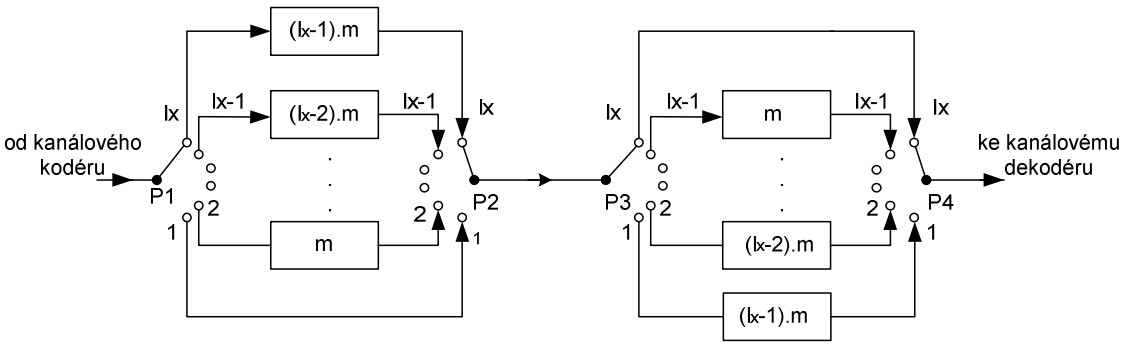
$$n \leq \frac{A+b}{l_x}. \tag{3.5}$$

Symboly, které jsou v kanále vedle sebe, jsou na výstupu vzdáleny o *D* a zpoždění D_t, které je zavedeno během prokládání je

$$D_t = l_x \cdot (l_x - 1). \tag{3.6}$$

Aby nedošlo k rozprostření kódového slova a tím případně k zasažení předchozím či následujícím shlukem chyb, je zapotřebí zajistit podmínku

$$A + b \geq l_x \cdot (l_x - 1). \tag{3.7}$$



Obr. 3.3: *Schématické znázornění konvolučního prokladače*

Tento systém skýtá jistou výhodu oproti bitovému prokladu, která se využívá např. u technologie ADSL [5]. Je totiž možné snadno měnit tzv. hloubku prokladu a tím operativně reagovat na dění v přenosovém kanále. Hloubka prokladu značí rozteč dvou původně po sobě jdoucích bitů za prokladačem. Z toho vyplývá, že čím větší je hloubka prokladu, tím více je

systém odolný vůči delším shlukům chyb. Zvyšování hloubky prokladu se děje přidáváním větví v systému prokladu.

3.3 Porovnání metod prokládání

V předchozím textu byli podrobně popsány techniky prokládání. Metoda bitového prokládání je nejjednodušší variantou prokladu. Kritickým se však stává zpoždění celého procesu. Je totiž zapotřebí vždy počkat, až se celá prokládací matice naplní, a poté je teprve možné vysílat proložená data. Jak je uvedeno v (3.1), zpoždění prokládacího procesu je rovno dvojnásobku velikosti prokládací matice, přičemž při použití kódu s nízkou korekční schopností narůstá právě její velikost.

Efektivnější se jeví prokládání konvoluční. Celý proces se děje kontinuálně, nemusíme tedy čekat na zaplnění všech paměťových buněk posuvných registrů jako je tomu u bitového prokladu. Proces je tedy rychlejší a zavádí se menší zpoždění. Z technického hlediska jsou však kladeny vyšší nároky na synchronizaci přepínačů. Tato metoda se s výhodou používá v systémech, kde se velmi často mění podmínky přenosu, např. v mobilních komunikacích. S každou prokládací metodou úzce souvisí použitý zabezpečovací kód, který ve své podstatě významnou měrou ovlivňuje parametry prokládacího systému. Tyto dvě entity jsou na sobě velmi úzce závislé a spolu tvoří systém, který má být schopen zabezpečit přenos proti shlukům chyb.

Výběr nejvhodnější metody pak závisí na konkrétním případě použití. Musíme brát v úvahu konkrétní distribuci chyb v kanále, konkrétní účel použití daného přenosového systému a požadavky na přenos, dále pak požadavky a možnosti pro realizaci systému. Zřetel je třeba brát nejen na technickou náročnost ale i na další parametry daných prokládacích a zabezpečovacích obvodů. Významným parametrem se pak stává celková doba zpoždění, kdy např. při přenosu hovorových signálů nemá být větší než cca 40ms.

4 ROZBOR MOŽNÝCH ŘESENÍ

Cílem diplomové práce je zabezpečit shluk chyb *b* maximální délky 64 bitů, který následuje vždy minimálně po $A = 640$ bezchybně přenesených bitech. Budeme tedy hledat možná řešení zadaného problému s tím, že do možných řešení se budeme snažit zařadit jak kódy, které splňují požadavek bezpečného přenosu, tak i kódy, jenž poskytnou více než základní požadavek.

Pro objektivní výběr optimálního řešení budeme jednotlivé varianty PKS hodnotit bodovým systémem, přičemž všechny zúčastněná hlediska budou mít stejnou váhu. Rozsah hodnocení bude od 0 po 100 bodů. Nejlepší variantě v dané kategorii výběru PKS bude přisouzeno právě 100 bodů, další varianty získají bodové hodnocení dle přímé úměry.

4.1 Varianta s bitovým prokládáním

V kap. 3.1 byl podrobně popsán princip bitového prokládání. Nyní se budeme zabývat možnými variantami kódů, které budeme kombinovat právě s bitovým prokladem. Mezi možné varianty bude zahrnut jednoduchý Hammingův kód, Bose-Chaudhuri-Hocquenghem kód a Fireův kód.

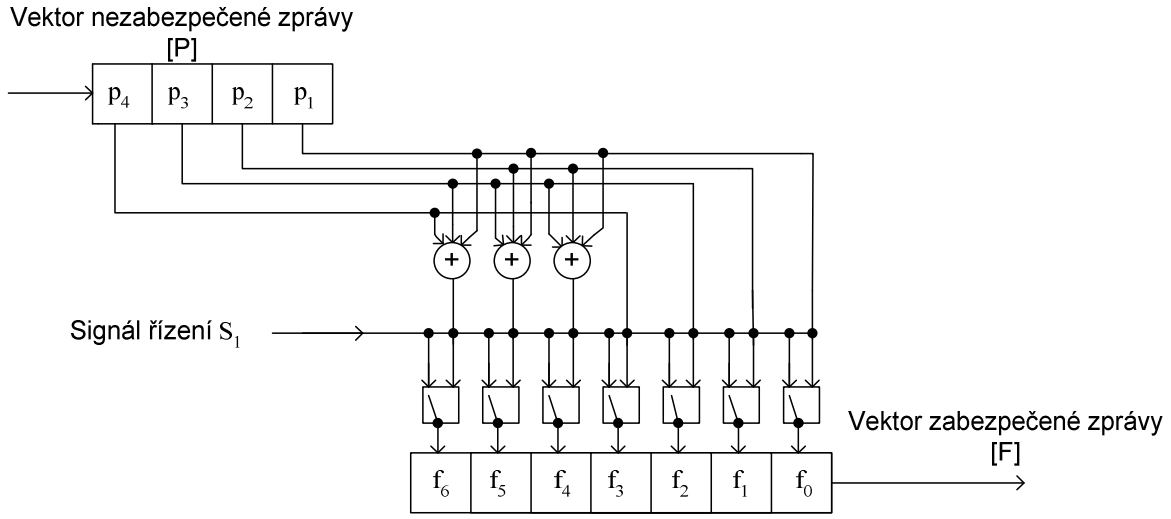
Do výběru by bylo možno zahrnout např. i často používaný Reed-Solomonův kód, který disponuje vysokým informačním poměrem a má také dobrou schopnost opravovat dlouhé shluky chyb. V našem případě je však zapotřebí opravit pouze shluk délky 64 bitů, což je relativně malé číslo, a tak by došlo k nevyužitelnosti robustnosti RS kódu, což spolu se složitosti a výpočetní náročnosti kódování a zejména dekódování zapříčiňuje ustoupení od jeho dalšího použití. Jako výhodnější varianta by mohlo být navržení RS kódu pro přímou opravu shluku chyb bez použití prokladače.

4.1.1 Hammingův kód

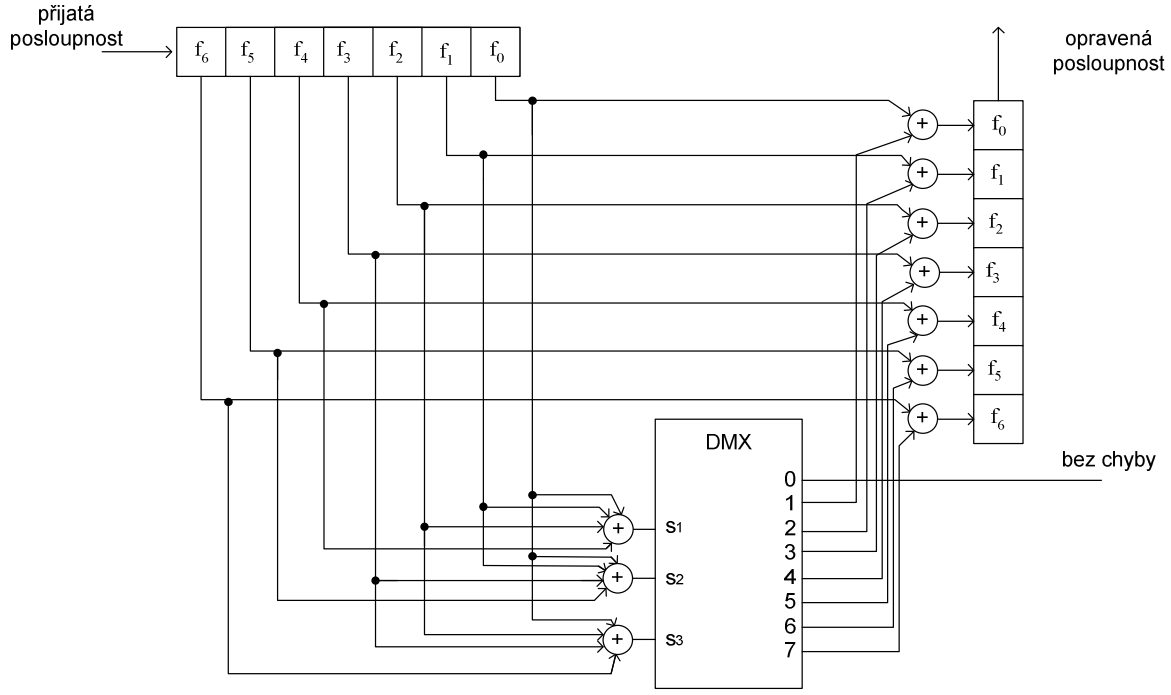
Jedná se o blokový korekční kód, jehož minimální Hammingova vzdálenost je $d_{min} = 3$. V případě Hammingova kódů pro smíšené zabezpečování dochází ke korekci dvou chyb, $d_{min} = 4$. Jeho hlavní výhodou je, že pomocí něj dokážeme opravit jednoduché chyby s co nejmenší myslitelnou redundancí [9].

Pro naše zadání použijeme základního Hammingova kódu (7 ; 4), který opravuje jednu chybu a budeme jej kombinovat s prokládáním. Bylo by možné například použít Hammingův kód (63 ; 57), který má informační rychlost přesahující 0,9, ale jeho vytvářecí i kontrolní matice by obsahovala velký počet řádku i sloupců a tak i realizace kodéru a dekodéru by byla

složitá. Vytvářecí matici [G] = [M_k;C_{rk}] a kontrolní matici [H] Hammingova kódu (7 ; 4) je možné nalézt v [2]. Z jejich tvaru poté vychází zapojení kodéru a dekodéru, *obr. 4.1, 4.2*.



Obr. 4.1: *Blokové schéma kodéru Hammingova kódu (7 ; 4)*



Obr. 4.2: *Blokové schéma dekodéru Hammingova kódu (7 ; 4)*

Rozměry prokládací matice jsou podle (3.1) a (3.2) následující

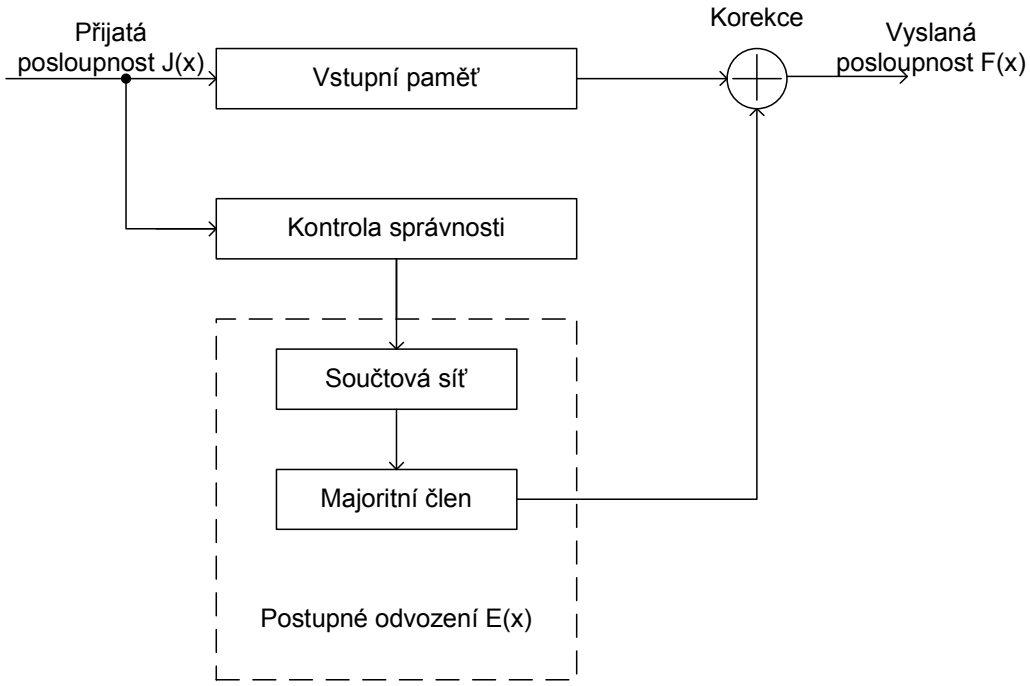
$$j \geq \frac{b}{t} \geq \frac{64}{1} \geq 64\,, \qquad n \leq \frac{A+b}{j} \leq \frac{640+64}{64} \leq 11\,.$$

Podmínka (3.2) je pro Hammingův kód (7 ; 4) splněna a tak rozměr prokládací matice bude $64 \times 7 = 448$ paměťových buněk. Pro další kódy nebude uváděn příklad výpočtu velikosti prokládací matice, vše je shrnuto v Tab. 4.1.

4.1.2 Bose – Chaudhuriho - Hocquenghemovy kódy

BCH kódy dle [10] patří do skupiny cyklických kódů, jenž pracují s daty v binární podobě. Binární BCH kód (*n*,*k*) umožňuje opravu *t* chyb a mají následující parametry:

- $n = 2^m - 1$ maximální délka kódového slova (4.1)
- $k \geq n - mt$ počet informačních bitů v kódovém slově (4.2)
- $d_{\min} \geq 2t + 1$ minimální Hammingova vzdálenost. (4.3)



Obr. 4.3: *Obecné blokové schéma obvodu pro majoritní dekódování BCH kódů*

Pro dekódování je možné použít výpočetní dekódování či majoritní dekódování, *obr. 4.3*. Pro naše zadání je možné použít dva kódy BCH a to kód opravující *t = 2* či *t = 3* chyby. Pro prvně zmíněnou variantu se nabízí BCH kód (*15 ; 7*), který díky svým parametrů právě dvě chyby dokáže opravit. Kód BCH (*15 ; 5*) se poté nabízí pro opravu tří chyb.

BCH kód (*15 ; 7*), je zadán vytvářecím mnohočlen $G(x) = x^8 + x^4 + x^2 + x + 1$. Z něj je možné podle [2] odvodit schéma kodéru, tedy děličky mod G(*x*) s přednásobením x^8. Pro dekódování se používá majoritní dekodér [10].

Kód BCH (*15 ; 5*) sloužící pro opravu $t = 3$ chyb, tedy s minimální Hammingovou vzdáleností $d_{min} = 7$. Vytvářecí mnohočlen G(x) bude dle BC teorému sestrojen ze součinu těch nerozložitelných mnohočlenů $g_i(x)$, jejichž $2t = 6$ kořenů Galoisova tělesa jde podle exponentů po sobě. Pro GF(2^4) lze podle [6] nalézt $G(x) = (x^4 + x^2 + 1)\ (x^4 + x^3 + x^2 + x + 1)\ (x^2 + x + 1) = x^{10} + x^8 + x^5 + x^4 + x^3 + x^2 + 1$.

Kodér bude opět tvořen děličkou mod G(x) s přednásobením x^{10}. Strukturu dekodéru je možné nalézt v [10]. Jelikož kód BCH (15 ; 5) má minimální Hammingovu vzdálenost $d_{min} = 7$, je třeba v kontrolní matici [H] tohoto kódu nalézt $d^* = d_{min} - 1 = 6$ kontrolních součtů. Podle [10] však tyto součty nelze nalézt v jednom kroku a tak je třeba použít majoritní dekodér se dvěma kroky majorizace.

4.1.3 Fireův kód

Fireův kód *(n,k)* je dle [3] blokový cyklický kód jehož vytvářecí mnohočlen je následujícího tvaru

$$G(x) = N(x) \,.\, (x^c + 1)\,, \tag{4.4}$$

kde je *N(x)* nerozložitelný mnohočlen řádu *m*, náležející exponentu *e*, při čemž platí, že *c* není dělitelné *e*.
Exponent *e* se dá vypočítat dle následujícího vztahu

$$e = 2^m - 1. \tag{4.5}$$

Parametry Fireova kódu jsou:

- délka kódové kombinace $n = e \,.\, c$ (4.6)
- počet zabezpečovacích prvků $r = c + m$ (4.7)
- počet zabezpečovaných prvků $k = n - r$ (4.8)

Fireův kód může pracovat ve 3 třech režimech – detekční mód, korekční mód a v tzv. smíšeném zabezpečení. Pro naše zadání použijeme korekčního módu pro který platí následující.

Řád mnohočlenu *N(x)* určuje délku zabezpečeného shluku chyb *b* v kódové kombinaci chyb *n* a zároveň musí platit

$$c \geq 2b - 1 \ , \ m \geq b \, . \qquad (4.9)$$

V úvahu budeme brát tři kódy opravující t = 3, 4, 5 chyb. Jako příklad návrhu kódu uvádím návrh Fireova kódu pro opravu $t = b = 3$ chyb. Návrh Fireova kódu pro opravu 4 a 5 chyb je analogický.

Podle vztahu (4.6) odvodíme koeficient c a m

$$c \geq 2b - 1 \geq 2 \cdot 3 - 1 \geq 5 \, ,$$

$$m \geq b \geq 3 \, .$$

Exponent e podle (4.2)

$$e = 2^m - 1 = 2^3 - 1 = 7.$$

Délka kódové kombinace, počet zabezpečovacích prvků a počet zabezpečených prvků podle (4.6), (4.7) a (4.8)

$$n = e \, . \, c = 7 \, . \, 5 = 35,$$

$$r = c + m = 5 + 3 = 8,$$

$$k = n - r = 35 - 8 = 27.$$

Navržený kód bude tedy F(35 ; 27), nerozložitelný mnohočlen $N(x)$ bude řádu 3. K nalezení přesného tvaru mnohočlenu použijeme tabulky nerozložitelných mnohočlenů $N(x)$ pro konstrukci Fireových kódů, které můžeme nalézt v [3]. Vybíráme polynom tvaru

$$N(x) = x^3 + x + 1 \, ,$$

druhý polynom bude dle vztahu (4.4) tvaru $x^5 + 1$. Potom vytvářecí mnohočlen Fireova kódu (35 ; 27) je

$$G(x) = x^8 + x^6 + x^5 + x^3 + x + 1.$$

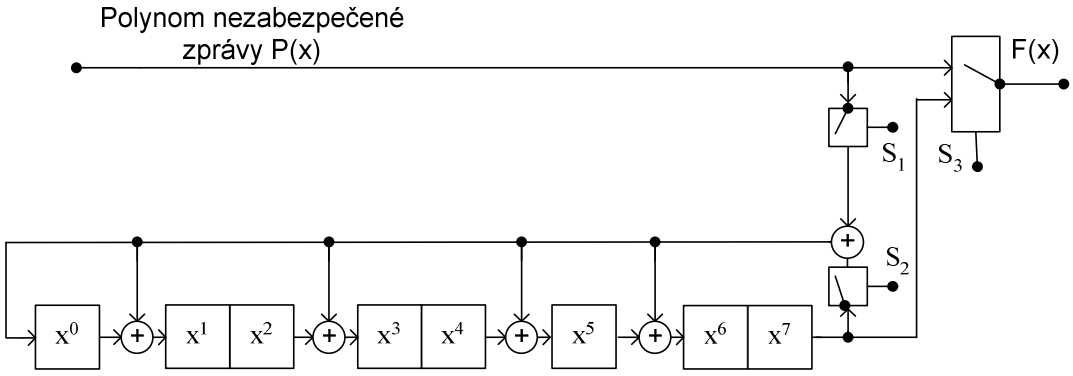
Nyní přistoupíme k návrhu prokládací matice. Opět vycházíme ze vztahů (3.1) a (3.2) následující

$$j \geq \frac{b}{t} \geq \frac{64}{3} \geq 22 \, ,$$

$$n \leq \frac{A + b}{j} \leq \frac{640 + 64}{22} \leq 32 \, .$$

Velikost prokládací matice bude $22 \times 32 = 704$. Narážíme zde však na problém. Požadovaná délka kódové kombinace pro prokládací matici jest menší nebo rovna 32, kódová

kombinace pro navržený Fireův kód je však 35. Jako řešení se nabízí tzv. zkrácení Fireova kódu. Jeho popis je možné naleznout v [3]. Princip zkracování délky kódové kombinace lze realizovat několika způsoby. První možností je přidání tolika nulových bitů před přijatou zkrácenou kódovou kombinaci, o kolik byla odvysílaná kódová kombinace zkrácena. Toto řešení je možné s výhodou využít při malé hodnotě zkrácení. Druhý způsob spočívá v předpokladu, že na prvních *i* místech nezkrácené kódové kombinace jsou nuly. V zapojení dekodéru poté musí docházet k úpravám, které právě vycházejí z tohoto předpokladu.



Obr. 4.4: *Kodér zkráceného Fireova kódu (32 ; 24) zadaného vytvářecím mnohočlenem* $G(x) = x^8 + x^6 + x^5 + x^3 + x + 1$

Potřebujeme tedy naleznout Fireův kód, který bude pracovat s bloky délky menší nebo rovno 32. Stupeň zkrácení *i* je zapotřebí volit s rozvahou. Musíme brát v úvahu, že při případné implementaci kodeku např. do mikročipu se lépe a efektivněji pracuje s bloky, které budou násobky osmi bitů. Jako nejmenší možné zkrácení *i* se nabízí hodnota 3, která ve výsledku dává 24 bitů jako nezabezpečenou posloupnost a 32 bitů jako zabezpečenou posloupnost. Obě tyto hodnoty jsou násobkem osmi a tak budeme uvažovat zkrácený Fireův kód (32 ; 24).

Kodér navrženého kódu F*(32 ; 24) vychází z vytvářecího mnohočlenu kódu F(35 ; 27). Jedná se o děličku mod $G(x)$ s přednásobením x^r, v našem případě x^8, *obr. 4.4*.

Jelikož stupeň zkrácení je roven pouze 3 bity, tak byl zvolen prvně zmíněný princip zkracování kódu, tedy zařazení tolika nulových bitů před přijatou zkrácenou kódovou kombinaci, o kolik byla odvysílaná kódová kombinace zkrácena. Při použití zkrácení dle druhé varianty by se celkové schéma a i princip dekódování značně zesložitil, [3].

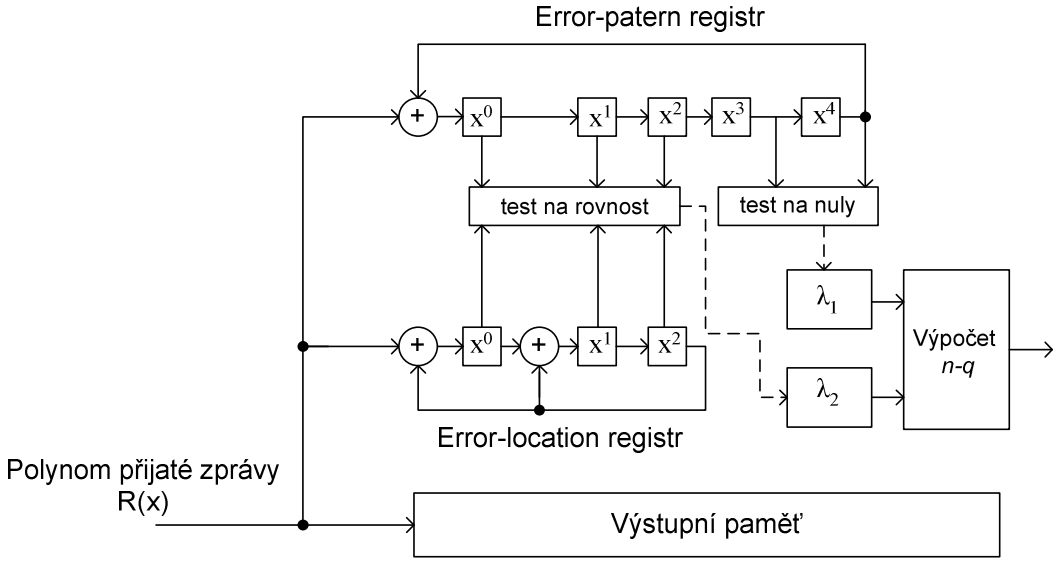
Princip zabezpečení pomocí cyklického kódu je následující. Do kodéru vstupuje polynom nezabezpečené zprávy P(x) a dělením této zprávy vytvářecím mnohočlenem G(x) se odvozují zabezpečovací bity R(x) jako zbytek po dělení, vztah (4.10).

$$\frac{P(x) * x^{(n-k)}}{G(x)} = M(x) + \frac{R(x)}{G(X)}, \qquad (4.10)$$

M(x) představuje mnohočlen podílu. Tento vztah se dále dá upravit pomocí algebry modulo 2, kdy operace sečítání odpovídá operaci sčítání na následující rovnici.

$$F(x) = P(x) * x^{(n-k)} + R(x) \tag{4.11}$$

Schéma dekodéru Fireova kódu je možné naleznout na *obr. 4.5*. Princip funkčnosti je následující, [7]. Do kodéru vstupují bloky dat délky 32 bitů, tyto je zapotřebí prodloužit třemi bity na délku 35 bitů. Je vypočten obsah registrů error-patern i error-location[1] jako zbytky po dělení mnohočlenu přijaté zprávy a mnohočlenů daných registrů. Nejdříve se ověřuje jakým druhem chyby byl daný blok napaden. Pokud je obsah obou registrů roven 0, tak k chybě nedošlo. Pokud jsou obsahy obou registrů nenulové, tak se jedná o opravitelnou chybu. Poslední možností je, že obsah jednoho registru je nulový a obsah druhého nenulový. Jedná se o překročení zabezpečovací schopnosti kódu a daná chyba nemůže být opravena.



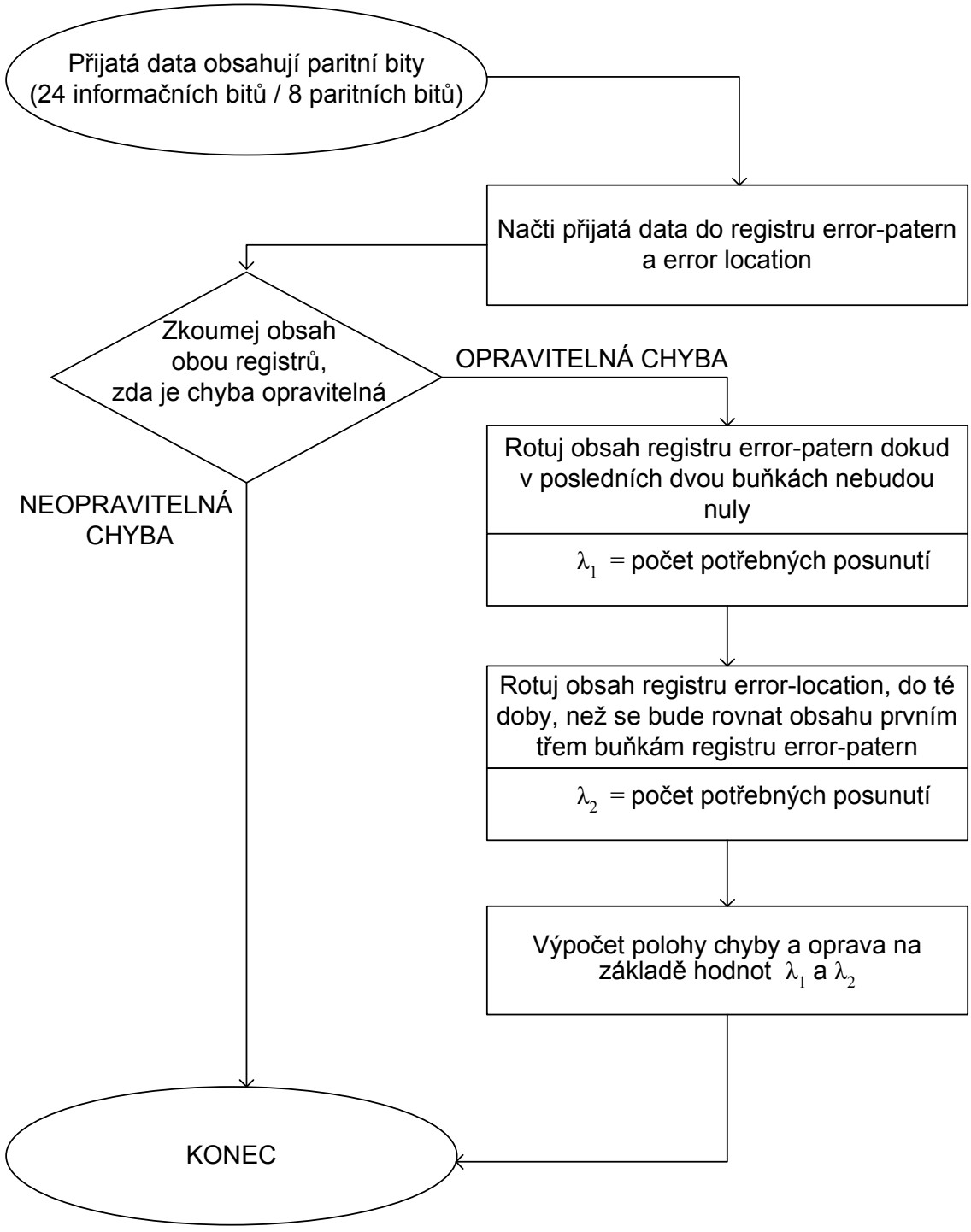
Obr. 4.5: *Dekodér zkráceného Fireova kódu (32 ; 24)*

Dalším krokem je rotace registru error-patern do té doby, než obsah posledních dvou paměťových buněk nebude roven 0. Počet rotací je roven hodnotě λ_1, přičemž musí platit $\lambda_1 < 2b - 2$.

Čtvrtý krok dekódování spočívá v rotaci obsahu registru error-location do té doby, než se jeho obsah bude rovnat obsahu prvních třech buněk registru error-patern. Počet rotací představuje proměnná λ_2. Opět je zde podmínka, že $\lambda_2 < e - 1$.

[1] Vzhledem k nejednoznačnosti v českém jazyce je v textu použito pro popis částí dekodéru anglických pojmů error-patern registr a error-location registr.

Posledním krokem se stává výpočet pozice a oprava chyby z hodnot λ_1 a λ_2, blíže lit. [7]. Pro názornost je uveden na *obr. 4.6* vývojový diagram procesu dekódování.



Obr. 4.6*: Vývojový diagram dekódování Fireova kódu*

4.1.4 Shrnutí PKS s bitovým prokládáním

Jako možné kódy pro použití s bitovým prokládáním byli brány v úvahu - Hammingův kód, dva BCH kódy a tři Fireovy kódy. Parametry celého PKS s daným kódem jsou uvedeny v Tab.4.1.

Tab. 4.1: Parametry PKS s bitovým prokládáním a jednotlivými kódy

kód	informační poměr	kodér	dekodér	prokládací matice	počet opravených chyb	stupeň zkrácení
	R	*PB*	*PB*	*PB*	*t*	*i*
Hamm (7 ; 4)	0,57	11	14	448	1	x
BCH (15 ; 7)	0,47	8	23	480	2	x
BCH (15 ; 5)	0,33	10	25	330	3	x
F*(32 ; 24)	0,75	8	40	704	3	3
F*(44 ; 33)	0,75	8	60	704	4	61
F*(54 ; 40)	0,74	8	66	702	5	225

Z předchozí tabulky je patrné, že hodnoty velikosti paměťových buněk kodéru a dekodéru jsou vůči velikosti prokládací matice zanedbatelné. Zaměříme se tedy na informační poměry kódů, složitost procesu dekódování a také na velikosti jednotlivých prokládacích matic s nimiž je úzce spojeno zpoždění, které se zavádí do celého zabezpečovacího procesu.

Prvním navrženým kódem byl Hammingův kód. Jedná se o kód s jednoduchým procesem dekódování a opraví pouze jednu chybu. Spolu s opravdu nízkou informační rychlostí se nestává vhodným kandidátem.

Dále byli brány v úvahu dva BCH kódy. Tyto kódy pracují s malými bloky dat a dokáží opravit 2 resp. 3 chyby. Díky tomu je velikost jejich prokládacích matic téměř poloviční ve srovnání s PKS s Fireovými kódy. Nevýhodou obou BCH kódu je, že pro opravu daného počtu chyb potřebuji téměř stejné množství nadbytečných bitů jako je bitů užitečných. V zadání práce je požadováno aby daný kód měl informační poměr větší nebo roven 0,5. Jelikož tuto podmínku ani jeden z kódů nesplňuje, tak oba BCH kódy z výběru vhodných variant odpadají.

U zbylých tří Fireových kódu přesahuje informační rychlost hodnotu 0,7. Velikosti jednotlivých prokládacích matic jsou prakticky totožné. Výběrovým kritériem se tedy stává stupeň zkrácení. Jak už bylo zmíněno výše, s rostoucím stupněm zkrácení roste i složitost dekodéru. Proto jako optimální kód byl vybrán Fireův kód F*(32 ; 24) se stupněm zkrácení $i = 3$. Výsledné hodnocení PKS s jednotlivými kódy je možno naleznout v Tab. 4.2.

Tab. 4.2: Výsledné hodnocení PKS s bitovým prokládáním

kód	informační poměr	prokládací matice	celkem	pořadí
	body	body	body	-
Hamm (7 ; 4)	76	74	150	4.
BCH (15 ; 7)	63	69	132	6.
BCH (15 ; 5)	44	100	144	5.
F*(32 ; 24)	100	57	157	1.
F*(44 ; 33)	100	57	157	2.
F*(54 ; 40)	99	58	157	3.

5 OVĚŘENÍ FUNKČNOSTI NAVRŽENÉHO PKS

Pro počáteční ověření funkčnosti navrženého PKS je vhodná realizace pomocí softwarového řešení. Jako vhodný simulační nástroj bylo zvoleno prostředí MATLAB v němž jsou dle předchozí teorie navrženy dílčí části PKS.

5.1 Návrh PKS v prostředí MATLAB

K dispozici máme verzi MATLAB 7.1. Celý PKS znázorněný na *obr. 5.3* byl zapsán do tzv. m-files, ve kterých je rozepsán algoritmus jednotlivých částí zabezpečovacího procesu. Bylo vytvořeno celkem šest těchto souborů s názvy PKS, koder, prokladani, chyby, zpetne_prokladani, dekodér.

5.1.1 M-file PKS

V souboru PKS se nalézá kostra celého programu. Definují se zde parametry kódu *n* a *k*, velikost délky shluku chyb (shluk_chyb), velikost délky ochranného intervalu (ochranny_interval) a délka vstupní posloupnosti (vstupni_stream).

```
%konstanty
b = 3; r = 8; e = 7; n = 32;
A1 = 3; A2 = -2;
konst1 = A1*(2*b-1);
konst2 = A2*e;

ochranny_interval = 640;
shluk_chyb = 64;
```

Následně je vytvořen vektor požadované délky představující data, která chceme přenášet. Tento obsahuje Matlabem náhodně vygenerovanou posloupnost 1 a 0.

```
delka_vst_posl = 2400; %zvoleny pocet bitu, nasobky 24
vstupni_stream = randint(1,delka_vst_posl);  % vygenerovany
vstupni vektor
```

Dále jsou zde volány všechny funkce pro zpracování vstupní posloupnosti, které odpovídají výše uvedeným názvům m-files. Nakonec je ověřováno, zda celý proces proběhl v pořádku, a zda byli všechny vzniklé chyby opraveny. Toto se provádí ověřením rovnosti

vektoru vstupni_stream a vektoru dekodovany_stream. Výsledek je znázorněno výpisem do konzoly, *obr. 5.1*.

```
%overeni zda prenos probehl v poradku
if vstupni_stream == dekodovany_stream
    disp('Data napadená shlukem chyb byla opravena.')
else
    disp('Nedošlo k opravě!!!')
end
```

5.1.2 M-file koder

V tomto m-file je realizován proces zabezpečení pomocí Fireova kódu (32 ; 24) Do kodéru vstupuje bitový tok v podobě vektoru vstupni_stream a z něj jsou postupně z každých k = 24 bitů odvozovány zabezpečovací prvky. Důležité je upozornit, že vstupní vektor musí být prodloužen na celkovou délku 27 bitů, aby mohl v podstatě vstupovat do kodéru nezkráceného Fireova kódu (35 ; 27). Výstupem z kodéru se tedy stává vektor délky 35 bitů, který má však na prvních třech pozicích uměle přidané nuly, které se mohou zanedbat. Výsledný zabezpečený tok je umístěn v proměnné zakodovany_stream.

```
vstup_r = zeros(1,length(vstup)+r);
vstup_r(1,1:27) = vstup;

g = [1,0,1,1,0,1,0,1,1]; %generujici mnohoclen G(x) = x^8 + x^6
+ x^5 + x^3 + x + 1

[M R] = deconv(vstup_r, g); %proces odvozeni zabezpecujich bitu
R = mod(R,2);

zabezpeceny = vstup_r + R; %vystup z koderu
zabezpeceny = zabezpeceny(1,4:35); %musime odstranit prvni 3
nuly aby byl vektor delky 32
```

5.1.3 M-File prokladani

Zde je s výhodou využito implementované funkce matintrlv, která realizuje bitové prokládání. Jako vstupní parametry jsou požadovány počet řádku a sloupců prokládací matice, které jsou dle teoretického odvození v předchozí kapitole rovny 22 a 32. Pro správné zaplnění prokládací matice je však nejdříve zapotřebí dorovnat zakodovany_stream do délky, která je násobkem velikosti prokládací matice. Toto se děje doplněním toku nulovými hodnotami, které je zapotřebí na přijímací straně následně odstranit. Výsledný proložený tok je uložen v proměnné prolozeny_stream.

```
%bitove prokladani dat
for i=1:704:length(zakodovany_stream)-703
    blocek = zakodovany_stream(i:i+704-1);
    prolozeny_stream = [prolozeny_stream, matintrlv(blocek, 22, 32)];
end
```

5.1.4 M-file chyby

Při realizaci chyby, je třeba brát v úvahu distribuci chyb, která je definována v zadání. Je tedy vytvořen vektor stream_chyb, který obsahuje vždy 640 nul, reprezentující bezchybný interval a poté 64 náhodně vygenerovaných bitů, které představují shluk chyb, *obr. 5.2*. Datový tok napadený shluky chyb chybovy_stream následně vznikne operací xor vektoru prolozeny_stream a stream_chyb.

```
%stream napadeny shluky chyb pri zachovani bezchybneho int
chybovy_stream = xor(stream_chyb,prolozeny_stream);
```

5.1.5 M-file zpetne_prokladani

Pro zpětné bitové prokládání je využito inverzní funkce k dopřednému prokladu, která se nazývá matdeintrlv. Výstupem se stává vektor deprolozeny_stream, ze kterého je však zapotřebí odstranit přebytečné nuly, které byly přidány před prokládáním.

```
%proces zpětného prokladani
for i=1:704:length(chybovy_stream)-703
    blocek2 = chybovy_stream(i:i+704-1);
    deprolozeny_stream =
[deprolozeny_stream,matdeintrlv(blocek2, 22, 32)];
end
```

5.1.6 M-file dekodér

Při dekódování je deprolozeny_stream rozdělen na posloupnosti 32 bitů. K těmto je opět zapotřebí uměle přidat 3 na dorovnání délky do 35 bitů. Tento blok vstupuje do dekodéru (35 ; 27). Nejdříve je vypočten obsah error-patern-registru s1, který je dále rotován do té doby, než obsah posledních dvou registrů není nulový. Počet rotací je uložen v proměnné lambda1. V tuto chvíli je odvozen obsah error-location-registru s2, který je následně rotován ve zpětnovazebném registru do té doby něž se jeho obsah nerovná obsahu prvních třech paměťových buněk v registru error-patern. Počet rotací představuje proměnná lambda2.

```
%definice registru
error_patern_register = [1, 0, 0, 0, 0, 1];
error_location_register = [1, 0, 1, 1];

%vypocet obsahu obou registru po jejich naplneni
[M s1] = deconv(pomocna3,error_patern_register);
s1 = mod(s1,2);
s1 = s1(1,31:35); %zajima nas jen poslednich 5bitu registru

[M s2] = deconv(pomocna3,error_location_register);
s2 = mod(s2,2);
s2 = s2(1,33:35); %zajima nas jen posledni 3bity reg
```

Z proměných lambda1 a lambda2 je odvozen lokátor chyby q.

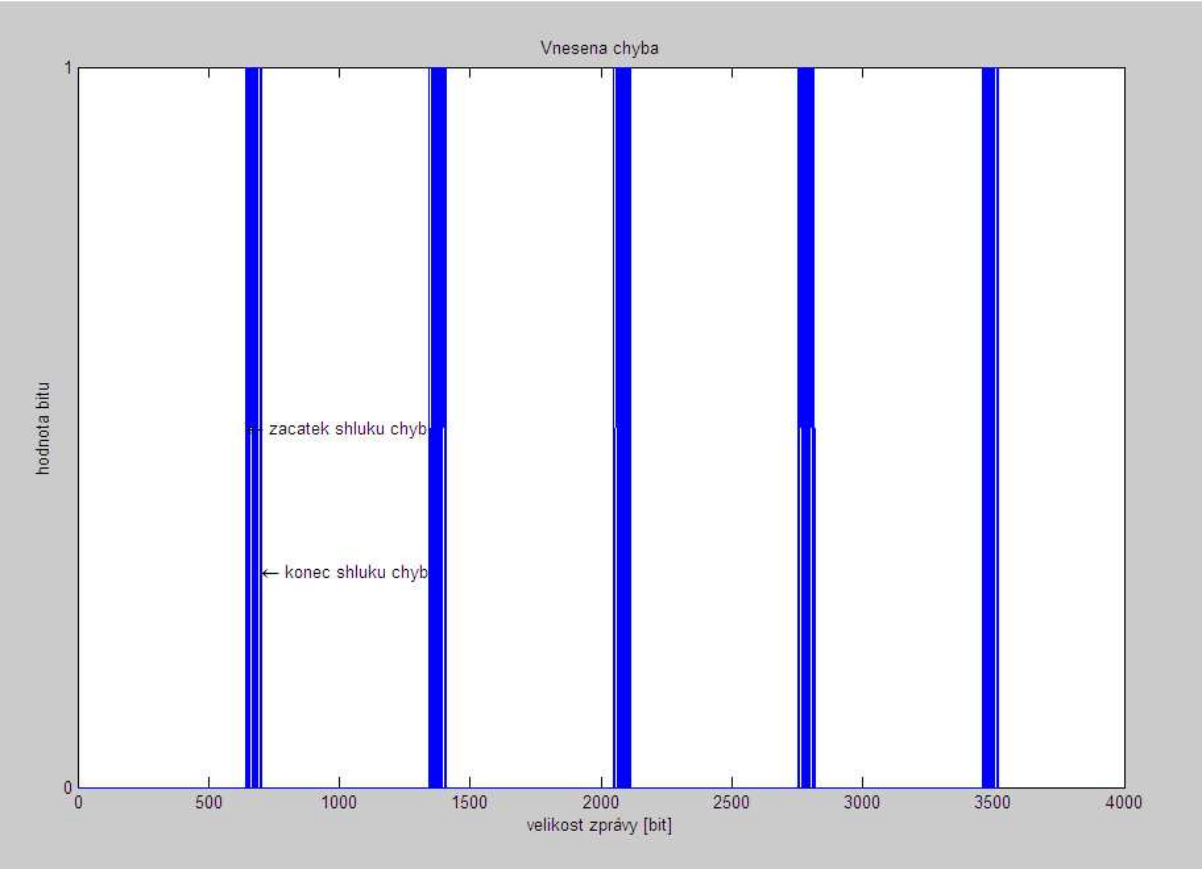
```
%vypocet q, zbytek po celociselnem deleni
citatel = konst1*lambda2 + konst2*lambda1;
jmenovatel = 35;
q = mod(citatel,jmenovatel);
```

K opravě chyby dochází operací xor chybového vektoru e a proměnné pomocna3 ve které je vždy uložen blok délky 35 bitů. Vektor dekodovany_stream obsahuje výstup z dekodéru. Pokud celý proces proběhl v pořádku, tento by se měl rovnat vektoru vstupni_stream.

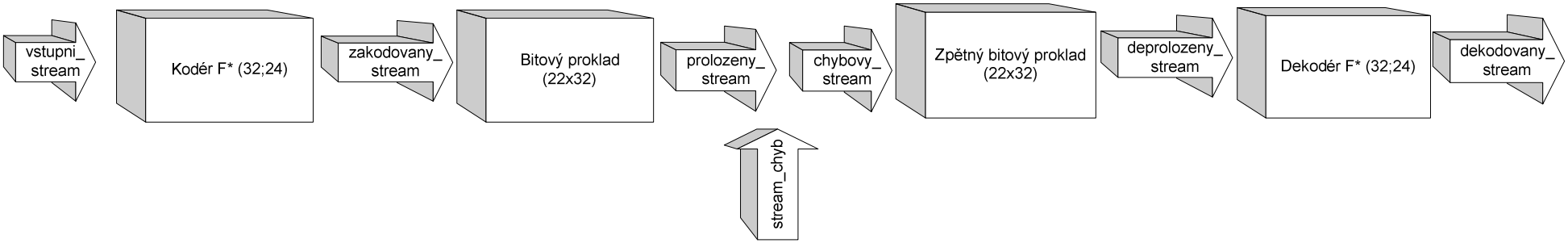
```
%%%oprava%%%
opraveny_blok = xor(pomocna3,e);
%odstraneni prvnich 3 nul
opraveny_blok = opraveny_blok(1,4:35);
```

```
PKS s bitovým prokládáním a zkráceným Fireovým kódem (32,24)
velikost ochranného intervalu:
   640

velikost shluku chyb:
    64

počet napadených bloků:
    86

data napadená shlukem chyb byla opravena
>>
```

Obr. 5.1*: Výstup z konzole v Matlabu*



Obr. 5.2: *Ukázka vygenerovaného shluku chyb*



Obr. 5.3: *Schématické znázornění PKS s proměnnými*

6 NÁVRH REALIZACE PKS

Dalším bodem zadání práce je vypracování návrhu realizace tak, aby byl výsledný systém schopen pracovat jako individuální. V současné době existují různé možnosti způsobu realizace protichybových systému a s těmito se seznámíme v následující kapitole. Bude vybrána optimální realizace a dále bude tato realizace navržena, popsána a ověřena.

6.1 Možné způsoby realizace PKS

Způsoby realizace můžeme rozdělit do dvou základních tříd, a to realizace hardwarová a realizace pomocí softwarového řešení. Dle lit. [11] se v současnosti pro hardwarovou realizaci používají následující tři řešení:

- Číslicové integrované obvody základních řad (74.., 4000, ECL 10 000, atd.),
- Mikrokontroléry a mikropočítače,
- Programovatelné logické obvody PLD (Programmable Logic Device) a obvody FPGA (Field Programmable Gate Arrays).

Číslicové integrované obvody základních řad

- Výhody – rychlá reakce, malá spotřeba při použití obvodů CMOS, snadná dostupnost.
- Nevýhody – malý stupeň integrace, cena, omezená možnost miniaturizace, při jakékoli úpravě nutnost změny plošného spoje.

Mikrokontoléry a mikropočítače

- Výhody – snadná cenová dostupnost, při změně funkčnosti kodeku zásah pouze do programu (jazyk C), ne do desky plošných spojů, možnost realizace složitých algoritmů, univerzálnost.
- Nevýhody – nižší rychlost reakce (problém zejména v systémech kde má procesor vykonávat řízení v reálném čase).

Programovatelné logické obvody

- Výhody – rychlost reakce, při změně funkčnosti kodeku zásah pouze do programu (jazyk HDL), snadná dostupnost vývojového prostředí (Xilinx, Altera),
- Nevýhody – menší univerzálnost oproti mikropočítačům.

Pro realizaci pomocí softwarového řešení je možné použít prostředí vyšších programovacích jazyků, např. C nebo C++. Tyto jsou běžně dostupné v podobě freeware nástroje Dev C++, nebo na školním serveru MSDNAA [13] v podobě nástroje Visual Studio.

Softwarové řešení jazykem C, C++

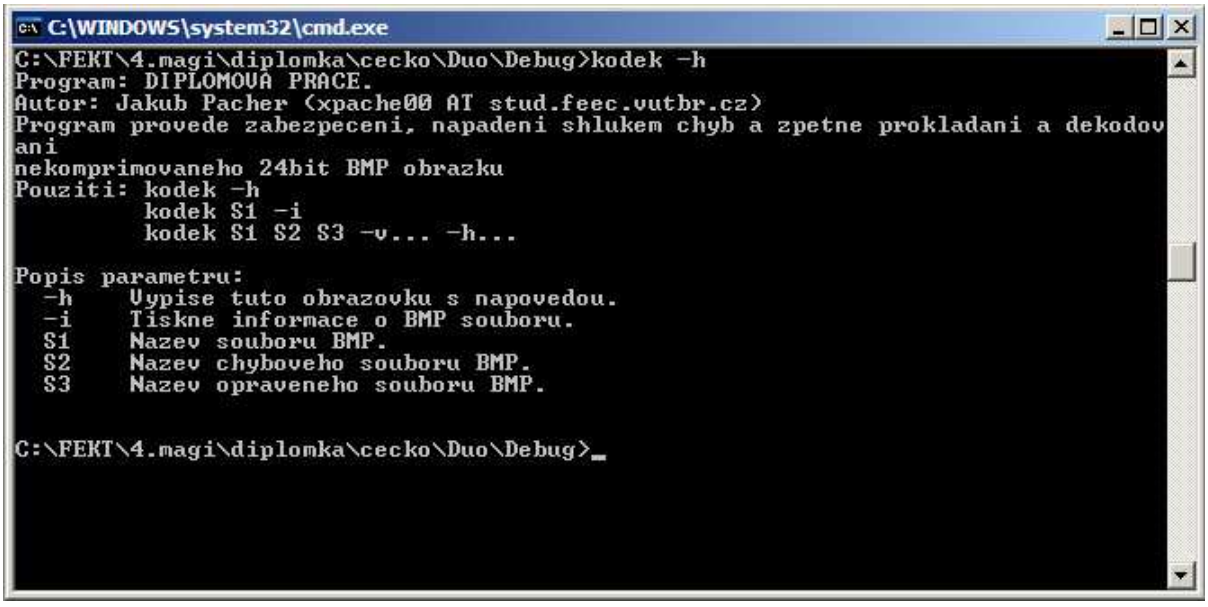
- Výhody – snadná úprava programu při testování, rychlost, snadná dostupnost vývojového prostředí, cena,
- Nevýhody – realizace na vyhrazeném PC.

Pro návrh realizace bylo zvoleno softwarové řešení pomocí jazyka C++ a to z těchto hlavních důvodů. S jazykem C++ má řešitel zkušenost, bude splněn požadavek na schopnost práce systému jako individuálního. Další výhodou se stává nízká cena a snadná možnost úpravy při vývoji.

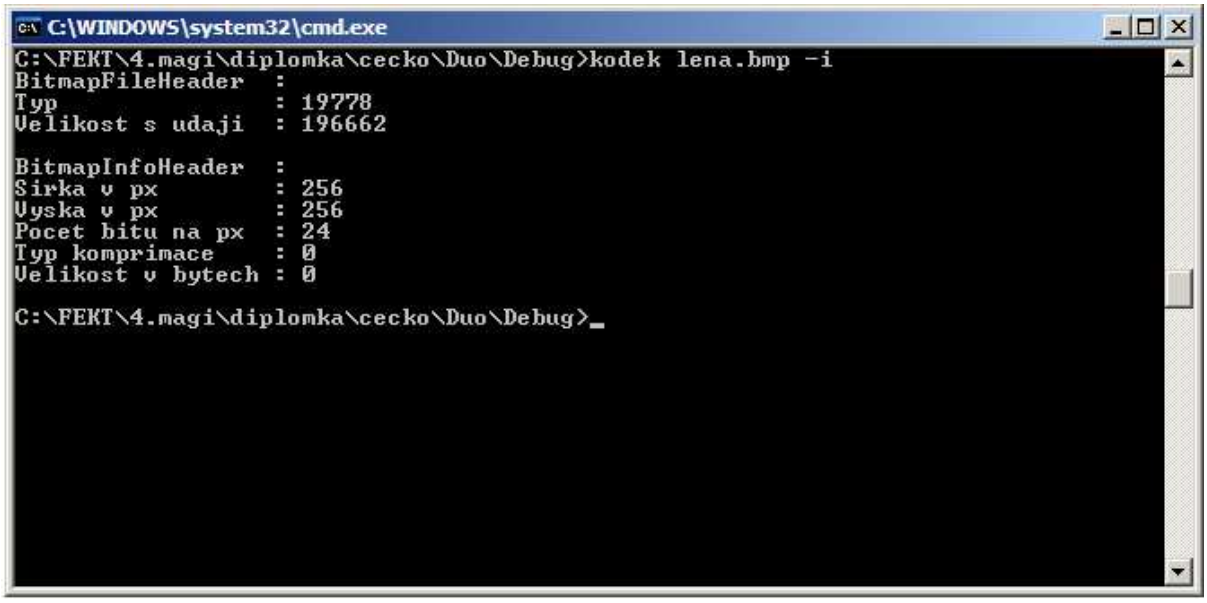
6.2 Realizace PKS v jazyce C++

Výsledná realizace PKS s prokládáním je vytvořena pomocí programovacího jazyka C++ a jako vývojové prostředí bylo použito nástroje Visual Studio 2008. Funkční aplikace je spustitelná na každém PC pomocí příkazové řádky a je určena pro přenos obrázků typu bitmapa.

Pro základní informace o aplikaci a způsob jejího ovládání je možno spustit nápovědu zadáním názvu spustitelného souboru s parametrem –h, *obr 6.1*. Základní informace o přenášeném souboru jako je jeho velikost, šířka a výška v pixelech, či počet bitů na pixel, je možno zjistit zadáním názvu spustitelného souboru a názvu bitmapového obrázku s parametrem –i, *obr. 6.2*.



Obr. 6.1: *Ukázka spuštění nápovědy*



Obr. 6.2: *Ukázka výpisu základních informací o přenášeném souboru*

6.2.1 Popis funkčnosti navrženého PKS

Zrealizovaná aplikace je navržena pro práci s rastrovými (obrazovými) soubory typu bitmapa (BMP). Tyto jsou nezávislé na daném zařízení (Device Independent Bitmap). Formát BMP je navržen tak, že umožňuje ukládání rastrových dat ve čtyřech formátech – 1bit, 4bit, 8bitů a 24bitů na pixel, [12]. S posledně zmíněným formátem, též zvaném TrueColor, pracuje navržený systém.

Hlavní program je uložen v souboru *main.cpp*, dále byl vytvořen soubor *hReadWrite.cpp* kde jsou funkce pro čtení a zápis přesného počtu bytů. K tomuto souboru je dále přidružen hlavičkový soubor s názvem *hReadWrite.h*, který funguje jako rozhraní pro funkce čtení a zápisu daného počtu bytů.

Aplikace se spouští zadáním názvu spustitelného souboru spolu se třemi parametry představující postupně název souboru S1, který chceme přenášet, název souboru, který bude obsahovat chyby S2 a název souboru S3, který bude opraven. Při spuštění aplikace je dále vytvořen binární soubor s předem definovaným názvem *prenos.bin*. Po provedení procesu zabezpečení je v tomto souboru obsažen vstupní bitmapový obrázek obohacený o zabezpečovací data odvozena z Fireova kódu a dále proložena bitovým prokladačem s hloubkou prokladu 22 bitů. Ve své podstatě je určen k přenosu v zadáním definovaném komunikačním kanálu. To znamená, že je zaručena oprava při maximálním shluku chyb délky 64 bitů vždy po 640 bezchybných bitech.

6.2.1.1 Definice globálních proměnných

Jak bylo zmíněno výše, ve funkci *main.cpp* je vytvořen hlavní program. Na začátku jsou definovány globální proměnné jako jsou generující mnohočlen kódu, mnohočlen location registru a patern registru. Dále zde máme inicializovány proměnné, které později slouží pro ukládání bloků dat z obrázku, z kodéru a dekodéru. V paměti jsou reprezentovány datovou strukturou jednorozměrné pole obsahující prvky typu unsigned integer (bezznaménkový integer). Např. definice error patern registru $x^5 + 1$ vypadá následovně

```
unsigned int patern_registr[] = {1, 0, 0, 0, 0, 1};.
```

6.2.1.2 Popis použitých struktur

Pro práci s bitmapovým obrázkem byli vytvořeny dvě struktury, *BitMapFileHeader*, která obsahuje základní informace o souboru a její velikost je 14 bytů, a struktura *BitMapInfo-Header*, která obsahuje základní metainformace o uloženém obraze. Její velikost je také konstantní a to 40 bytů. Dále byli definovány struktury pro zpracování parametrů z konzole *params*, a struktura pro práci s prokládací maticí *deinterleaveUnit*. Např. definice struktury *BitMapFileHeader* vypadá následovně.

```
typedef struct BitMapFileHeader
  {
  //definice jednotlivych casti struktury
  } TBitMapFileHeader;.
```

6.2.1.3 Popis práce s obrázkem

Program nejdříve načte soubor S1 (matici RGB obrázku) a z něj vytvoří pole *RGBArray2D*. Je přitom použito funkce *rgbToInt()*, která převádí prvky z matice RGB na proměnné typu integer. Pole *RGBArray2D* je následně po blocích 24 bitů podrobeno procesu kódování funkcí *encode()* a vzniká tak zabezpečený obrázek v poli *RGBEncode2D*. Princip funkčnosti kodéru je popsán v kap. 4.1.3. V běžícím programu je reprezentován následujícím kódem.

```
void encode(unsigned int *p)
   {
   //proces kodovani -> p(x)*x^(n-k)/g(x)
   }
```

Obsah pole *RGBEncode2D* je následně proložen bitovým prokladačem. Prokládací matice je definována jako dvourozměrné pole velikosti 22 řádku a 32 sloupců pomocí následujícího příkazu.

```
unsigned int interleaveArray[22][32];
```

Výsledek procesu prokládání se ukládá do binárního souboru *prenos.bin*, který je určený k přenosu. Další část programu již představuje stranu přijímače. Zde v prvním kroku dochází k načtení „přijatého“ souboru *prenos.bin*.

```
if ((frprenos = fopen("prenos.bin", "rb")) == NULL)
 {
  printMessage(EBADFILE);
  exit(1);
 }
```

Pro názornost a zároveň i pro ověření funkčnosti celého systému jsou do přijatého souboru zavedeny shluky chyb. K tomuto účelu byla vytvořena funkce *attack()*, která napadá každou prokládací matici shlukem chyb 64 bitů a to tak, že mění hodnoty log. 1 na log. 0 a naopak. Tímto je zaveden nejhorší možný případ shluku chyb. Na řadu přichází operace zpětného prokládání, která je inverzní k operaci dopředného prokladu a výsledek tohoto procesu se ukládá do pole *RGBDekode2D*. Pro představu, jak by vypadal obrázek napadený shluky chyb ještě před procesem dekódování, je vytvořen soubor S2. Jeho název definuje uživatel a obsahuje degradovaný obrázek vlivem přenosu přes komunikační kanál.

Následuje operace dekódování, která vychází z vývojového diagramu uvedeného na *obr.4.5*. Postupně se tedy z pole *RGBDekode2D* odebírají bloky velikosti 32 bitů pro něž se volá funkce *repair()* představující dekodér. Výsledný opravený obrázek je uložen do souboru s názvem S3.

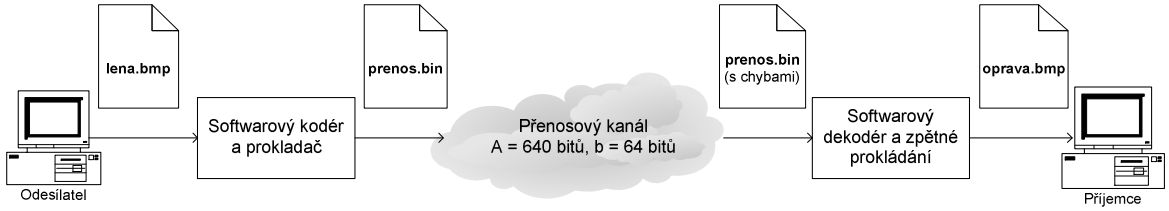
```
void repair(unsigned int *p)
   {
   //proces dekodovani
   }
```

6.2.2 Ukázka funkčnosti navrženého PKS

Nyní bude předvedena názorná ukázka funkčnosti celého systému. Pro tento účel je do souboru určeného pro přenos, prenos.bin, uměle vložena chyba, která představuje zadáním definovaný přenosový kanál. Jako testovací obrázek byl zvolen počítačovými grafiky oblíbený obrázek Lena. Jedná se o obrázek s rozlišením 256x256 pixelů ve formátu TrueColor. Po spuštění aplikace s požadovanými parametry, *obr. 6.3*, jsou do složky, ve které se nalézá spustitelný soubor, vygenerovány následující soubory.

Binární soubor prenos.bin, BMP soubor chyba.bmp, *obr. 6.5*, a BMP soubor oprava.bmp, *obr. 6.6*. Jak bylo popsáno výše, binární soubor obsahuje zabezpečená a proložená data z obrázku lena.bmp. Po průchodu přenosovým kanálem definovaným v zadání by soubor lena.bmp vypadal tak jak je uvedeno na *obr. 6.5*. Zde je možno vidět pruhy, které vznikly díky periodicky se opakujícímu shluku chyb délky 64 bitů. Jelikož je použito zabezpečení spolu s prokládáním tak je tento chybový soubor opraven PKS a získáváme opět původní obrázek, který byl vyslán odesílatelem. Tímto je ověřena funkčnost celého systému, který je zobrazen na *obr. 6.4*.

Obr. 6.3: *Ukázka spuštění PKS s požadovanými parametry*



Obr. 6.4: *Znázornění celkového přenosového systému*

Obr. 6.5: *BMP soubor chyba.bmp*

Obr. 6.6: *BMP soubor oprava.bmp*

ZÁVĚR

Tato práce se zabývá protichybovými kódovými systémy s prokládáním, které kombinují techniku zabezpečovacího kódování s technikou umožňující rozložit dlouhé shluky chyb na kratší shluky či na chyby samostatné. Na začátku se čtenář seznámí s možnými druhy chyb a s principem protichybového kódování. Pro přehlednost je na závěr první kapitoly uveden přehled a rozdělení používaných zabezpečovacích kódů.

V další kapitole je rozebrána metoda prokládání. Její hlavní princip a účel a také její možné varianty. Těmi jsou bitové a konvoluční prokládání. Výhoda bitového prokládání je v její jednoduché realizaci, naopak nevýhodou se stává zpoždění, které do celého zabezpečovacího cyklu vkládá. Konvoluční prokládání je naopak variantou prokladu s mnohem menším zpožděním. Nevýhodou celého systému je ale jeho náročnost na synchronizaci. Ze zadání vyplývá, že ve výsledném navrženém systému má být použito bitového prokládáni a tak se varianta s konvolučním prokladem již neobjevuje.

Ve čtvrté kapitole se již přistupuje k samotnému návrhu protichybového systému s prokládáním. Jsou zde podrobně rozebrány tři možné zabezpečovací kódy a jejich varianty. Na základě stanovených kritérií a požadavků v zadání je vybrán nejvhodnější systém, kterým se stal protichybový kódový systém s bitovým prokládáním a zkráceným Fireovým kódem F*(32,24).

Ze zadání diplomové práce vyplývá, že má být ověřena funkčnost navrženého systému. Pro počáteční ověření funkčnosti je vybráno prostředí MATLAB. Je zde vytvořena skupina m-files představující jednotlivé části protichybového systému a to koder, prokladani, chyby, zpetne_prokladani a dekoder. Zastřešující m-file nazvaný PKS generuje množinu náhodných dat, volá jednotlivé části PKS a poté ověřuje zda přenos proběhl v pořádku.

Posledním požadavkem v zadání bylo vypracování návrhu realizace PKS. V závěrečné kapitole se čtenář nejdříve seznámí s možnými variantami realizace PKS, které se v současné době používají. Jsou zde shrnuty jejich výhody a nevýhody a je vybrána realizace, která je pro konkrétní systém nejvýhodnější. Touto se stala softwarová realizace za použití jazyka C++. Byla vytvořena aplikace pracující s bitmapovými obrázky, kdy po načtení požadovaného obrázku je vytvořen binární soubor, který obsahuje zabezpečená a proložená data. Takto zabezpečený soubor je určen pro přenos v zadáním definovaném kanálu. Na přijímací straně dochází k načtení přijatého souboru a pokud obsahuje chyby tak dochází na základě použité zabezpečovací techniky i k jeho opravě. Tímto považuji zadaní diplomové práce za splněné v celém rozsahu.

SEZNAM LITERATURY

[1] SHANNON, C.E. *A Mathematical Theory of Communication* .Bell Syst. tech. J., Vol. 27, No. 3, July 1984 , pp. 379-423, and Vol. 27, No. 4, October 1948, pp. 623-656.

[2] NĚMEC, K. *Datová komunikace*. Skriptum. Ústav telekomunikací VUT v Brně, 1998, ISBN 80-7204-088-X.

[3] NĚMEC, K. *Protichybové kódové systémy zabezpečující proti dlouhým shlukům chyb* [online]. 2005 [cit. 2010-02-08]. poslední revize Dostupné z WWW: <http://www.elektrorevue.cz/clanky/06034/>

[4] ŽALUD, V. *Moderní radioelektronika*. BEN-technická literatura, Praha 2004. 656s. ISBN 80-86056-47-3 / 9788086056470

[5] *Asymmetric digital subscriber line (ADSL) transceivers : ITU-T Recommendation G.992.1*., 1999. 255 s.

[6] HOUGHTON, A. *Error Coding For Engineers*. Kluwer Academic Publishers, 2001. 246 s. ISBN 0-7923-7522-X.

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

[8] HANUS, S. *Bezdrátové a mobilní komunikace.*. Skriptum. Ústav radioelektroniky VUT v Brně, 2005, ISBN 80-214-1833-8.

[9] ADÁMEK, J. *Kódování*. Nakladatelství technické literatury, Praha 1989.

[10] NĚMEC, K. *Protichybové kódové zabezpečení s BCH kódy* [online]. 2005 [cit. 2010-02-10]. Dostupné z WWW: <http://www.elektrorevue.cz/clanky/05015/>

[11] KOLOUCH, J.: *Programovatelné logické obvody a návrh jejich aplikací v jazyku VHDL*. [Skriptum FEKT VUT v Brně.] Brno, 2007.

[12] TIŠŇOVSKÝ, P.: *Grafický formát BMP* [online]. 2006 [cit. 2010-04-18]. Dostupné z WWW: <http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblibeny/>

[13] TRÁVNÍČEK, Jiří. *Informační miniportál* [online]. 2010 [cit. 2010-04-20]. MSDN Academic Alliance. Dostupné z WWW: <http://msdnaa.feec.vutbr.cz/>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ADSL	-	Asymmetric Digital Subsriber Line
ARQ	-	Automatic Repeat Request
BCH	-	Bose-Chaudhuri-Hocquenghem
BER	-	Bit Error Rate
BMP	-	Bitmapa
FEC	-	Forward Error Correction
FPGA	-	Field Programmable Gate Array
GF	-	Galois Field
GSM	-	Global Systém for Mobile Communication
LDPC	-	Low Density Parity Check
PKS	-	Protichybový kódový systém
PLD	-	Programmable Logic Device

A	úsek bezchybně přenesených bitů
b	délka shluku chyb v bitech
d	Hammingova vzdálenost
D_t	zpoždění vznikající prokladem
d_{min}	minimální Hammingova vzdálenost
$F(x)$	mnohočlen zabezpečené zprávy
$\mathbf{G}$	vytvářecí matice kódu
$G(x)$	vytvářecí mnohočlen kódu
$\mathbf{H}$	kontrolní matice kódu
i	stupeň zkrácení Fireova kódu
j	počet řádků prokládací matice
$J(x)$	mnohočlen přijaté zprávy
k	délka nezabezpečeného bloku
l_x	počet větví prokladače
n	délka zabezpečeného bloku
$N(x)$	nerozložitelný mnohočlen kódu
$P(x)$	mnohočlen zabezpečené zprávy
R	informační rychlost kódu
S	celková složitost kódu
t_1	počet detekovaných chyb
t_2	počet korigovaných chyb
Z	celkové zpoždění způsobené kódem

A PŘÍLOHY

A.1 Obsah přiloženého CD

Název souboru/složky	Popis
xpache00.pdf	text diplomové práce ve formátu .pdf
metadata.pdf	metadata k diplomové práci
Kodek.exe	spustitelný soubor pro zabezpečení BMP obrázku
lena.bmp	BMP obrázek určený k přenosu
PKS s Fireovým kódem (32,24)	složka obsahující zdrojové kódy pro MATLAB
Kodek	složka obsahující projekt v C++