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

# SOFTWARE PRO PODPORU VÝUKY KOREKČNÍCH KÓDŮ

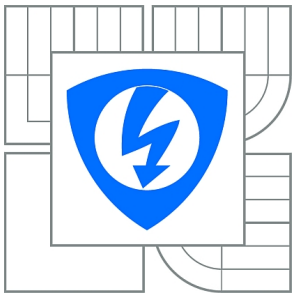
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JÁN PETRIK
AUTHOR

BRNO 2014


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


# SOFTWARE PRO PODPORU VÝUKY KOREKČNÍCH KÓDŮ

SOFTWARE FOR LEARNING SUPPORT OF ERROR-CORRECTING CODES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS


AUTOR PRÁCE — JÁN PETRIK

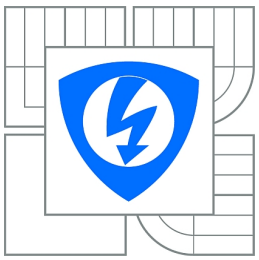
AUTHOR

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

SUPERVISOR



BRNO 2014



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Bakalářská práce

bakalářský studijní obor
**Teleinformatika**

| | | | |
|---|---|---|---|
| ***Student:*** | Ján Petrik | ***ID:*** | 147427 |
| ***Ročník:*** | 3 | ***Akademický rok:*** | 2013/2014 |

**NÁZEV TÉMATU:**

## Software pro podporu výuky korekčních kódů

**POKYNY PRO VYPRACOVÁNÍ:**

Nastudujte a popište problematiku korekčních kódů. Na tomto základě navrhněte koncept řešení softwarové podpory výuky vybraných korekčních kódů. Při návrhu konceptu věnujte pozornost pedagogickým aspektům jako je interaktivita a názornost. Výukový software prakticky zrealizujte, otestujte a zpracujte pro něj uživatelský manuál.

**DOPORUČENÁ LITERATURA:**

[1] Burda K.: Bezpečnost informačních systémů. VUT v Brně, Brno, 2013.
[2] Matoušek R.: Metody kódování. VUT v Brně, Brno, 2006.

| | | | |
|---|---|---|---|
| ***Termín zadání:*** | 10.2.2014 | ***Termín odevzdání:*** | 4.6.2014 |

***Vedoucí práce:*** doc. Ing. Karel Burda, CSc.
***Konzultanti bakalářské práce:***

**doc. Ing. Jiří Mišurec, CSc.**
*Předseda oborové rady*

**UPOZORNĚNÍ:**

## ABSTRAKT

Tato Bakalářská práce se zaměřuje na vybrané korekční kódy (ECC) a jejich problematiku. Cílem práce je pomoci čtenářům pochopit ECC snadnou a srozumitelnou formou. Najdeme zde mnoho názorných tabulek a obrázků pro lepší pochopení jak kódy pracují. každá sekce obsahuje jeden kód a většinou je naplněna způsobem zabezpečení a principem dekódování. Pro každou sekci byl vytvořen interaktivní aplet a program pro experimentální účely.

## KLÍČOVÁ SLOVA

korekční kódy, Opakovací kód, rozšířený Hammingův kód, Cyklický kód, Cyklický redundantní součet, Konvoluční kód, blokový kód, Aplet, ECC program.

## ABSTRACT

This Bachelor thesis is mainly focused on issues of the chosen error correcting codes (ECC) and their demonstrative. Goal of this work is help to understand ECC to readers, easy and reasonable form. We can find here a lot of illustrative tables and figures for better understanding how codes works. Every section constains one code and mostly is filled by code securing and code decoding. For every section interactive Aplet and program for experimental uses were created.

## KEYWORDS

error correcting codes, Repeating code, Extended Hamming's code, Cyclic code, Cyclic redundancy check, Convolution code, block code, Aplet, ECC program.

PETRIK, Ján *Software pro podporu výuky korekčních kódů*: bakalářská práce. Místo: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 48 s. Vedoucí práce byl prof. Ing. Karel Burda, CSc.

# PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Software pro podporu výuky korekčních kódů“ jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Místo ..... .....

(podpis autora)

# PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Karlu Burdovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.


Místo ..... .....

(podpis autora)

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# 1 ÚVOD

Korekční kódy mají schopnost opravit jednotlivé bitové chyby při přenosu dat přes nespolehlivý nebo zašuměný komunikační kanál. Hlavní myšlenkou je, že odesílatel zakóduje zprávu s rendundancí za použití korekčního kódu. Prvním objevitelem byl americký matematik Richard Hamming, jenž v roce 1950 vynalezl první korekční kód: Hammingův (7,4) kód. Následně se tento kód rozšířil a tak vznikl Rozšířený Hammingův kód, který firma IBM použila ve svém prvním superpočítači IBM 7030.

Redundance dovolí přijímači opravit nebo detekovat omezený počet chyb v jakémkoli místě zprávy. Díky korekčnímu kódování je přijímač schopen opravit chyby sám a tím pádem nemusí žádat o opakované přeposlání zprávy. Nevýhodou však je snížení přenosové rychlosti, kvůli vložené rendundanci.

Aplikuje se v situacích, kde žádost o opakované přeposlání zprávy je velice nákladná nebo neuskutečnitelná. Toto kódování má pole působnosti v mnoha vědních oblastech, například se často přidává do velkokapacitních zařízeních pro obnovu zničených dat, modemů a najdeme jej i v počítačových pamětech.

V této práci se budeme zabývat vybranými typy korekčních kódu, které budou vysvětleny srozumitelnou formou. Postupně budeme přecházet od blokových ke konvolučním korekčním kódům. Výstupem práce je aplet ve formátu exe, psaný v objektově orientovaném programovacím jazyce ActionScript 3.0, který je součástí programu Adobe Flash CS5. Program je rozdělen podle sekcí v této práci. Následně je vytvořena webová stránka, taktéž rozdělena do již zmíněných sekcí, kde si zájemci mohou tento aplet spustit a navíc zde je i program psaný v jazyce C++, který bude sloužit k vyzkoušení si, jestli danou problematiku pochopili. Na tvorbu grafického prostředí bude z velké části používán Adobe Illustrator.

Applet je rozdělen na části jak je uvedeno v textu, v tomto pořadí: Opakovací kód, Hammingův (7,4) kód , Rozšířený Hammingův kód, Cyklický kód a konvoluční kód.

# 2 DŮLEŽITÉ POJMY

Zde si uvedeme základní pojmy, které jsou potřebné pochopení následujících kapitol:

## 2.1. Blokový kód

Má přesně určené rozložení informačních a zabezpečovacích míst v kódové kombinaci. Každá kódová kombinace systematického kódu o délce $n$, obsahuje $k$ informačních bitů a $m = n - k$ zabezpečovacích prvků, mluvíme o ($n$, $k$) kódu. Vždy platí že $n > k$.

## 2.2. Konvoluční kód

Patří mezi spojité kódy. Nerozdělují se na bloky a kódování se děje trvale tak, jak postupují informační prvky do kodéru. Posloupnost konvolučního kódu obsahuje obyčejně prokládané informační a kontrolní bity, přičemž kontrolní prvky se tvoří jako funkce informačních prvků (většinou ve tvaru lineární operace – posuvu a součtu modulo 2).

## 2.3. Kódový poměr a kódový zisk

Kódový poměr se určuje jako poměr informačních prvku vůči celkové délce slova:

$$R = \frac{k}{n} \tag{2.1}$$

Hammingův(7,4) kódový poměr se tedy spočítá $R = \frac{4}{7} = 0.57$.

Kódový zisk se obecně počítá $\frac{SNR_{nekod}}{SNR_{kod}}$, ale konkrétně u hammingova kódu: [1]

$$k\acute{o}dov\acute{y}\ zisk = (T_c + 1).R\ d\boldsymbol{B} \tag{2.2}$$

## 2.4. Hammingova vzdálenost

Počet bitů ve kterých se dvě kódová slova liší:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |

$d_{min} = 4$

## 2.5. Hammingova váha

Počet nenulových bitů v kódovém slově:

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |

$w = 4$

## 2.6. Minimální vzdálenost Hammingova kódu

Nejmenší počet bitů ve kterých se kódová slova liší:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $u_1$ | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| $u_2$ | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| $u_3$ | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| $u_4$ | 0 | 0 | 0 | 1 | 1 | 1 | 0 |

$d_{minG} = 2$

## 2.7. Korekční a detekční schopnost kódu

Tabulka 2.1 Shrnuje možnosti korekčních a detekčních kódů založených na minimální vzdálenosti. [6]

Například je-li minimální vzdálenost 5, kód je schopen buďto opravit 1bit a zároveň tři detekovat, nebo žádný neopravit, ale detekovat 4.

Tab. 2.1: Maximální schopnost opravit nebo detekovat bit

| **Minimální vzdálenost** | **korekce** | **detekce** |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0 | 1 |
| 3 | 1 | 1 |
| 4 | 1 | 2 |
| 5 | 2 | 2 |
| 6 | 2 | 3 |
| 7 | 3 | 3 |
| 8 | 3 | 4 |
| $d$ | $\frac{d-1}{2}$ | $\frac{d}{2}$ |

## 2.8. Plotkinova hranice

Uvádí potřebnou délku $n$ a počet zabezpečovacích prvků $m$.

Její matematické vyjádření je následující:

$$n \geq \frac{F.d_{min} - 1}{F - 1} \quad r \geq \frac{F.d_{min} - 1}{F - 1} - (1 + log_F.d_{min}) \tag{2.3}$$

kde F je počet stavů signálu. Obvykle je F=2 a pak se vztahy zjednoduší:

$$n \geq 2.d_{min} - 1, \quad r \geq \frac{F.d_{min} - 1}{F - 1} - (1 + log_F.d_{min}), \quad r \geq log_2 \frac{2^{2(d_{min}-1)}}{d_{min}} \tag{2.4}$$

**Příklad:** Nalezněte rozměry binárního kódu ($n$ ; $k$), který dokáže opravit 2 chyby při délce zabezpečované posloupnosti k=4.

Nejprve si určíme potřebnou Hammingovu vzdálenost: $d_{min} = 2d_1 + 1 = 2.2 + 1 = 5$

Poté určíme $n$ a $k$ z Plotkinovi hranice: [5]

$$n \geq 2.d_{min} - 1 \qquad r \geq log_2 \frac{2^{2(d_{min}-1)}}{d_{min}}$$

$$n \geq 2.5 - 1 \qquad r \geq log_2 \frac{2^{2(5-1)}}{5}$$

$$n \geq 9 \qquad r \geq 5,68$$

$n = k + m = 4 + 6 = 10$, což vyhovuje první nerovnosti. Hledaný rozměr (10 ; 4).

# 3 OPAKOVACÍ KÓD

Uvažujeme blokový kód v ideálním přenosovém kanále o $n = 2$, tudíž máme $2^n = 4$ možných kombinací kódových slov a těmi jsou:

| 00 | 01 | 10 | 11 |
|---|---|---|---|

Pokud bychom vysílali po blocích jak je uvedeno nahoře nebyl by dekodér nikdy schopen určit ani opravit chybu. Využijeme tedy opakovacího kódu a všechna kódová slova po sobě dvakrát zopakujeme. Získáme tyto bloky:

| 000000 | 010101 | 101010 | 111111 |
|---|---|---|---|

Nyní jsme rozšířili Hammingovu vzdálenost z d$min = 1$ na d$min = 3$. To nám umožní opravit právě jednu chybu viz 2.7. Tento kód je založen na vysoké pravděpodobnosti, kdy při změně jednoho bitu bude stále jednoznačné opravit chybný bit. Přijme-li dekodér chybnou zprávu například $\boxed{0\ 1\ 1\ 1\ 0\ 1}$, přijímač ví, že správné kódové slovo je $\boxed{0\ 1\ 0\ 1\ 0\ 1}$, jelikož byl zaměněn jen jeden bit správného kódového slova, naopak k ostatním by bylo zapotřebí zaměnit o mnoho více bitů, což je nepravděpodobné.

| Přijaté slovo | 011101 | 011101 | 011101 | 011101 |
|---|---|---|---|---|
| Možné kombinace | 000000 | 010101 | 101010 | 111111 |

Shrnutí:

- Výhodou tohoto kódu je, že se 3x zvýší spolehlivost přenosu dat.
- Nevýhodou je, že se o 1/3 sníží přenosová rychlost kvůli redundanci a také jsou známi mnohem efektivnější korekční metody.

Nyní si ukážeme příklad detekce chyby v opakovacím kódu, na to nám bude stačit přidat do původního kódového slova jen jeden nadbytečný bit, který bude fungovat pro kontrolu sudé parity. Tedy:

| 00 | 01 | 10 | 11 |
|---|---|---|---|
| 000 | 011 | 101 | 110 |

Sečteme-li operací XOR jednotlivý blok, vyjde nám 0 tzn. sudá parita. Tentokrát je d$min = 2$.

Uvedeme si příklad, kdy přijímač přijal posloupnost $\boxed{0\ 1\ 0}$. Kromě kódového slova $\boxed{1\ 0\ 1}$, se všechny ostatní kódy liší jen jedním bitem. Je tedy zřejmé, že chyba nastala, ale není možné zjistit přesně kde a chybu opravit.

| Přijaté slovo | 0 1 0 | 0 1 0 | 0 1 0 | 0 1 0 |
|---|---|---|---|---|
| Možné kombinace | 0 0 0 | 0 1 1 | 1 0 1 | 1 1 0 |

# 4 HAMMINGŮV (7,4) KÓD

Hammingův kód je lineární blokový kód pro opravu nebo detekci chyb pojmenovaný po jeho objeviteli Richardu Hammingovi. Předpokládejme, že přenášená data se skládají z informačních vektorových bitů $\boldsymbol{k}$, ke kterým se přidá zabezpečující vektorový bit $\boldsymbol{p}$. Konkrétně $\boldsymbol{p}$ je interpretováno jako celé číslo, které musí být v rozsahu od 0 do $m + k$, kde je $m + k + 1$ odlišných hodnot. Protože $m$ bit dokáže rozpoznat $2^m$ případů, musí platit Hammingovo pravidlo [6]

$$2^m = m + k + 1 \tag{4.1}$$

Zde si uvedeme názvosloví, které budeme v této a příští kapitole používat.

| | |
|---|---|
| $k$ | Počet informačních bitů. |
| $m$ | Počet kontrolních bitů |
| $n$ | Délka kódového slova, $n = m + k$. |
| $\boldsymbol{u}$ | Informační bitový vektor, |
| $\boldsymbol{p}$ | Kontrolní bitový vektor. |
| $\boldsymbol{s}$ | Vektor syndromu. |
| $t_d$ | Detekční schopnost kódu2.7 |
| $t_c$ | Korekční schopnost kódu2.7 |
| $d_{min}$ | Hammingova vzdálenost2.4 |
| $w$ | Hammingova váha2.5 |
| $d_{minG}$ | Minimální Hammingova vzdálenost kódu2.6 |

Všechny přenesené bity musí být zkontrolovány. Zabezpečovací bity musí být rozprostřeny mezi bity informačními viz Tabulka 4.1. Tento kód dokáže opravit max. 1 chybu.

Máme-li tři kontrolní bity , kolik dokážeme zabezpečit informačních bitů?
$m = 3$. Aplikujeme Hammingovo pravidlo (4.1).

$$\begin{aligned} 2^m &= m + k + 1 \\ 2^3 &= 3 + k + 1 \\ k &= 4 \end{aligned}$$

Délka kódového slova n = 7, kód (7,4)

## 4.1. Princip zabezpečení

V Tabulce 4.1 máme tři kontrolní bity, $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$. Tyto bity sledují výslednou paritu. Je-li $\oplus$ všech bitů ve zprávě roven nule poté je přenos zprávy bezchybný, naopak je-li $\oplus$ všech bitů roven 1, v přenosu nastala chyba.

Například přijímač přijme bitovou posloupnost, vyjádřenou decimálním číslem 11:

| $p_0$ | $p_1$ | $u_3$ | $p_2$ | $u_2$ | $u_1$ | $u_0$ |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |

Dekodér počítá výslednou paritu: $0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0$ (dekodér přijme zprávu jako bezchybnou)

Tentokrát změníme hodnotu informačního bitu $\mathbf{u}_1$ z 1 na 0.

| $p_0$ | $p_1$ | $u_3$ | $p_2$ | $u_2$ | $u_1$ | $u_0$ |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |

Dekodér počítá výslednou paritu: $0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$ (dekodér přijme zprávu jako chybnou a pokusí se zprávu opravit)

**Důkaz:** v tabulce 4.1 vidíme paritní bit $\mathbf{p}_0$ sloužící ke kontrole liché parity ($\mathbf{u}_0 = 1$).
Pro $\mathbf{p}_1$ ($\mathbf{u}_1 = 1$) a pro poslední paritní bit $\mathbf{p}_2$ ($\mathbf{u}_2 = 1$).
Proběhne tedy tento výpočet:

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

Výsledná posloupnost je tedy 1 1 0 = 6, chyba nastala v šestém místě a přijímač tento chybný bit opraví.

Tab. 4.1: Hammingův (7,4) kód

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|
| **informace** | $p_0$ | $p_1$ | $u_3$ | $p_2$ | $u_2$ | $u_1$ | $u_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 11 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 12 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 13 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 14 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Kód s vlastnostmi, kdy platí rovnost Hammingovy vzdálenosti 2.4 se nazývá Perfektní kód. Existuje pro $k = 1,4,11,26,57,120\ldots$

# 5 ROZŠÍŘENÝ(8,4) HAMMINGŮV KÓD

Vznikne rozšířením Hammingova(7,4) kódu o jeden paritní bit, který kontroluje paritu celkového kódového slova. Je schopný opravit jednu chybu a navíc detekovat dvě chyby. Princip je znázorněn v tabulce 5.1. Jaké události mohou v kódu nastat si podrobněji popíšeme. [6]

1. V Kódovém slově nejsou chyby tudíž je celková parita sudá a syndrom z $(n-1)$ je nulový.
2. V kódovém slově se vyskytl jeden chybný bit, celková parita je lichá. Mohou nastat dvě situace.
   - Chybný bit je umístěn v místě kontroly celkové parity, tudíž syndrom bude nulový, protože tento bit nijak neovlivňuje správnost informačního toku.
   - Chybný bit je umístěn jinde než v místě celkové parity, tudíž bude syndrom nenulový a indikuje chybný bit.
3. V kódovém slově se vyskytly dva chybné bity, celková parita bude sudá. Mohou nastat dvě situace.
   - Jeden ze dvou chybných bitů je umístěn v kontrole celkové parity, syndrom bude nenulový a indikuje chybný bit.
   - Dva chybné bity jsou, umístěny jinde než v místě celkové parity, syndrom bude nenulový, kódové slovo nelze opravit.

Tab. 5.1: Rozšířený Hammingův kód

| **Možnosti** | | | |
|---|---|---|---|
| **Počet chyb** | **Výsledná parita** | **Syndrom** | **Vyhodnocení přijímače** |
| 0 | sudá | 0 | bez chyby |
| 1 | lichá | 0 | Chyba v celkové paritě |
| 1 | lichá | $\neq 0$ | Syndrom indikuje chybný bit |
| 2 | sudá | $\neq 0$ | Dvojitý error (neopravitelné) |

Tab. 5.2: Hammingův (8,4) kód

| **informace** | **1** $p_0$ | **2** $p_1$ | **3** $u_3$ | **4** $p_2$ | **5** $u_2$ | **6** $u_1$ | **7** $u_0$ | **8** $p_3$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 3 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 8 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 11 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 12 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 13 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 14 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Z tabulky 5.2 je zřejmé, že nově přidaný paritní bit $p_3$, slouží ke kontrole sudé parity.

# 6 CYKLICKÝ KÓD

Cyklický kód je lineární blokový kód $(n, k)$ u kterých jsou ve vytvářecí matici G jednotlivé řádky obsahující stejné prvky, jen posunuty o jedno místo v jednom směru. Tyto kódy jsou zadány tzv. generujícím mnohočlenem $G(\mathrm{x})$. Podle řádu mnohočlenu je určen počet zabezpečovacích prvků $m = (n - k)$. Cyklické kódy mají v kódové kombinaci o délce $n$ prvků na prvních $k$ místech prvky nezabezpečené zprávy a na zbývajících $m$ místech zabezpečovací prvky [2].

Názvosloví které budeme v této kapitole používat [5]:

| | |
|---|---|
| $P(x)$ | Mnohočlen nezabezpečené zprávy |
| $G(x)$ | Generující mnohočlen |
| $R(x)$ | Mnohočlen zbytku |
| $F(x)$ | Mnohočlen zabezpečené zprávy |
| $S(x)$ | Mnohočlen syndromu |
| $J(x)$ | Mnohočlen přenesené zprávy |

16-ti stupňový polynom má 17bitů! Například polynom $x^5 + x^2 + x^3 + 1$ je vyjádřen binární hodnotou. $\boxed{1\ 0\ 0\ 1\ 0\ 1}$, nejvyšší polynom je sice $x^5$, ale nesmíme zapomenout na nejmenší bit, který není polynomicky vyjádřen.

## 6.1. Princip zabezpečení

Způsob zabezpečení můžeme matematicky popsat následovně [5]:

$$\frac{P(x).x^{n-k}}{G(x)} = M(x).\frac{R(x)}{G(x)} \tag{6.1}$$

Protože používáme dvoustavový signál tak $F = 2$ a pro ten platí pravidla algebry mod2, kde operace součtu se rovná operaci rozdílu, můžeme nezabezpečený blok $F(x)$ vyjádřit zápisem:

$$F(x) = P(x).x^{n-k} + R(x) = M(x).G(x) \tag{6.2}$$

Příklad: Máme nezabezpečený cyklický kód (15,7) zadaný $G(x) = x^7 + x^4 + x^2 + 1$. Potřebujeme zabezpečit zprávu $P(x) = x^6 + x^3 + x^2 + 1$.

| $G(x)$ | $P(x)$ |
|---|---|
| 1 0 0 1 0 1 0 1 | 1 0 0 1 1 0 1 |

Existují dva způsoby výpočtu zabezpečení zprávy:

1. Polynomické vyjádření.
2. Metoda postupného sčítání.

**1)** Vynásobíme mnohočlen nezabezpečené zprávy mnohočlenem zabezpečovacích $(n-k)$ prvků:

$$P(x) \,.\, x^{n-k} = (x^6 + x^3 + x^2 + 1) \,.\, x^8$$
$$P(x) \,.\, x^{n-k} = x^{14} + x^{11} + x^{10} + x^8$$

Výsledný mnohočlen vydělíme generujícím mnohočlene $G(x)$:

$$\begin{array}{l}
(x^{14} + x^{11} + x^{10} + x^8) : (x^7 + x^4 + x^2 + 1) = x^7 + x^3 + x^2 + x \\
\underline{x^{14} + x^{11} + x^9 + x^7} \\
\qquad x^{10} + x^9 + x^8 + x^7 \\
\qquad \underline{x^{10} + x^7 + x^5 + x^3} \\
\qquad\qquad x^9 + x^8 + x^5 + x^3 \\
\qquad\qquad \underline{x^9 + x^6 + x^4 + x^2} \\
\qquad\qquad\qquad x^8 + x^6 + x^5 + x^4 + x^3 + x^2 \\
\qquad\qquad\qquad \underline{x^8 + x^5 + x^3 + x} \\
\qquad\qquad\qquad\qquad \underline{\underline{x^6 + x^4 + x^2 + x}}
\end{array}$$

Podrobnější postup:

Nejvyšší polynom $P(x) \,.\, x^{n-k}$ se vydělí nejvyšším mnohočlenem $G(x)$ tzn.:

$$x^{14} : x^7 = x^7$$

Tento mnohočlen se zapíše do výsledku a dále se provede zpětné násobení $G(x)$, tedy:

$$x^7 \,.\, (x^7 + x^4 + x^2 + 1) = x^{14} + x^{11} + x^9 + x^7$$

Výsledek zapíšeme pod rovnici a provedeme $\oplus$ dvou pod sebe zapsaných polynomů:

$$(x^{14} + x^{11} + x^{10} + x^8) \oplus (x^{14} + x^{11} + x^9 + x^7) = x^{10} + x^9 + x^8 + x^7$$

Výsledek zapíšeme pod rovnici a dělíme ho nejvyšším mnohočlenem $G(x)$:

$$x^{10} : x^7 = x^3$$

Tento mnohočlen se zapíše do výsledku a dále se provede zpětné násobení $G(x)$, tedy:

$$x^3 \,.\, (x^7 + x^4 + x^2 + 1) = x^{10} + x^7 + x^5 + x^3$$

Výsledek zapíšeme pod rovnici a provedeme $\oplus$ dvou pod sebe zapsaných polynomů:

$$(x^{10} + x^9 + x^8 + x^7) \oplus (x^{10} + x^7 + x^5 + x^3) = x^8 + x^6 + x^5 + x^4 + x^3 + x^2$$

Tímto způsobem pokračujeme dokud nebude mnohočlen $G(x)$ vyšší než mnohočlen $P(x) \,.\, x^{n-k}$
Získáme tak mnohočlen zbytku $R(x)$, ve kterém se ukrývá zabezpečení zprávy. V tomto případě $R(x) = x^6 + x^4 + x^2 + x$

Hledané kódové slovo je tedy podle vztahu 6.2 $x^{14} + x^{11} + x^{10} + x^8 + x^6 + x^4 + x^2 + x$

**2)** Mnohočleny si vyjádříme binárně:

$$P(x) = 1001101$$
$$G(x) = 10010101$$

Ke zprávě připíšeme $(n-k)$ nul $(15-7=8)$. nezabezpečená zpráva je tedy v tomto tvaru: 100110100000000.
Operací $\oplus$ postupně sčítáme prvky pod sebou.

Prvky generující matice vždy začínají v místě, kde začíná binární 1 ve zprávě.

Sčítaní skončí, když už není dál co sčítat.

## 6.2. Detekce a korekce chyb

Detekce a korekce chyb se provádí tak, že přijatá zabezpečená zpráva se vydělí generujícím mnohočlenem $G(x)$.


Obr. 6.1: Metoda postupného sčítání

- $R(x) = 0$ ve zprávě se nevyskytla chyba
- $R(x) \neq 0$ ve zprávě se vyskytla chyba

Pro opravu malých kódů můžeme použít tabulku syndromů.

V souvislosti s Hammingovou vzdáleností musíme sestavovat kód tak, abychom nepřekročili jeho detekční nebo korekční schopnost. Uvedeme si příklad cyklického kódu (7;4) s postupem řešení.

Příklad: Mějme nezabezpečenou zprávu cyklického kódu (7;4) danou:

$$P(x) = x^2 + x + 1$$
$$G(x) = x^3 + x + 1$$

Zprávu zabezpečíme podle návodu v podsekci princip zabezpečení 6.1.
Zabezpečený mnohočlen $F(x) = x^5 + x^4 + x^3 + x$.

V přenosu dojde k chybě a přijímáč získá $J(x) = x^5 + x^4 + x^2 + x^3 + x$.

Přijímač začne zprávu dekódovat a to tak, že podělí $J(x) : G(x)$

$$(x^5 + x^4 + x^3 + x^2 + x) : (x^3 + x + 1) = x^2 + x$$
$$\underline{x^5 + x^3 + x^2}$$
$$\quad x^4 + x$$
$$\quad \underline{x^4 + x^2 + x}$$
$$\quad\quad \underline{\underline{x^2}}$$

V tomto bodě víme, že chyba nastala v místě $x^2$ (binárně $\boxed{1\ 0\ 0}$). Chyba byla detekována. K opravě tohoto mnohočlenu přijímač potřebuje znát tabulku syndromů

6.1. Podle tabulky zjistí, že posloupnost 1 0 0 je přiřazena k mnohočlenu $x^2$ a ten pomocí operace mod2 přičte k přenesené zprávě $J(x)$ :

$$(x^5 + x^4 + x^3 + x^2 + x) + (x^2) = x^5 + x^4 + x^3 + x$$

Dále přijímač vybere z mnohočlenu první ($k = 4$) prvky. Mohl by tedy vybrat $(x^6, x^5, x^4, x^3)$, ale do této množiny spadají z mnohočlenu $J(x)$ jenom $(x^5, x^4, x^3)$. Posledním krokem je vrácení mnohočlenu do základní polohy a to se provede vydělením $j(x)$ nejvyšším polynomem $G(x)$.

$(x^5 + x^4 + x^3) : (x^3) = \underline{\underline{x^2 + x + 1}}$
$\underline{x^5}$
$x^4 + x^3$
$\underline{x^4}$
$x^3$

Tímto přijímač opravil a dekódoval původní nezabezpečenou zprávu $P(x) = \boxed{1\ 1\ 1}$. Tabulka syndromů je však pro velké cyklické kódy neaplikovatelná, kvůli velikosti tabulky kterou by musel přijímač mít, proto používáme jiné techniky.

Tab. 6.1: Tabulka syndromů

| Chybný bit | syndrom | polynom syndromu |
|---|---|---|
| 0000000 | 000 | 0 |
| 1000000 | 100 | 1 |
| 0100000 | 010 | $x$ |
| 0010000 | 001 | $x^2$ |
| 0001000 | 110 | $1 + x$ |
| 0000100 | 011 | $x + x^2$ |
| 0000010 | 111 | $1 + x + x^2$ |
| 0000010 | 101 | $1 + x^2$ |

# 7 CYKLICKÝ REDUNDANTNÍ SOUČET (CRC)

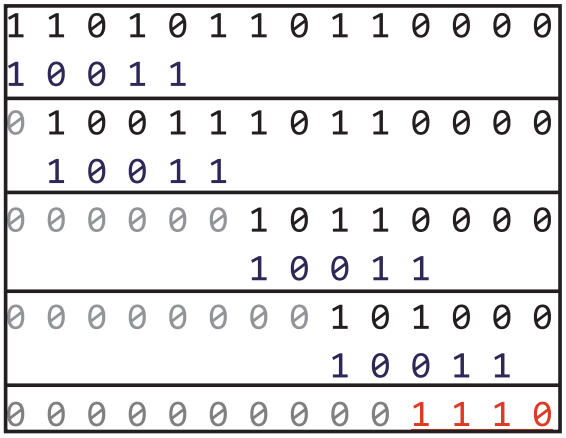
Princip spočívá v tom, že vysílač vykonstruuje pomocí funkce jistou hodnotu zvanou kontrolní součet (checksum), a tu připojí ke zprávě. Přijímač poté použije stejnou funkci k výpočtu kontrolního součtu přijaté zprávy a porovná ji s připojeným kontrolním součtem. aby zjistil zda byla zpráva správně přijata.

CRC se zpravidla liší ve své délce bloku. Delší bloky poskytují vyšší zabezpečení dat a jsou použitelné pro přenos velkých objemů dat. Na úkor toho produkují větší zbytkové hodnoty, což zvyšuje počet cyklů kontroly chyb. [4]

16-ti stupňový polynom jako je například CRC-CCITT ($x^{16} + x^{12} + x^5 + 1$) má 16-ti bitový zbytek.

CRC jsou klasifikovány podle nejvyšší nenulové hodnoty, která by měla být vždy 1.

Dobře zkonstruovaný CRC s danou velikostí bloku, je schopen detekovat jakýkoli lichý součet chyb v bloku a dva bity kdekoli v bloku.
U 16-ti stupňového CRC je 99,998% šance na detekci všech možných chyb ($2^{16}/2^{16} + 1$). [4]




Obr. 7.1: Výpočet kontrolního součtu

Přijímač může:

1. Oddělit zprávu od kontrolního součtu. Vypočítá přijatý kontrolní součet zprávy a porovná jej se součtem s připnutým kontrolním součtem.
2. Spočítá celé kódové slovo a zjistí, zda se rovná nule.

# 8 KONVOLUČNÍ KÓDY

Jsou jedny z nejpopulárnějších forem binárních korekčních kódů. Aplikují se v bezdrátových komunikacích (IMT-2000, GSM, IS-95), satelitních komunikacích, všesměrových systémech a v mnoha dalších odvětvích. Nejvíce dodnes používaný dekódovací algoritmus tzv. Viterbiho algoritmus, je založen na konvolučním dekódování. Při správné kombinaci s prokládáním v káskádovém schématu se jejich účinnost přibližuje Shannonovu limitu. kódování může probíhat dvěma způsoby buďto generující maticí, nebo generujícími mnohočleny.

Základní jednotkou je konvoluční kodér, jenž disponuje pamětí v tom smyslu, že výstupní stav nezávisí jen na symbolech které do kodéru vstupují, ale i na jeho předchozích stavech [3].

## 8.1. Funkce konvolučního kodéru

Na obrázku je znázorněn konvoluční kodér disponující jedním vstupem ($u_1$) a dvěma výstupy ($v_1, v_2$). Tento kodér je znázorněn na obrázku 8.1. Stavová tabulka kodéru je uvedena v tabulce 8.1. Pro určení kódového poměru musíme znát [5]:

Informační rychlost

$$R = u_0/v_0$$

Délka kódového ohraničení

$$v = m \,.\, k_0$$

Udává jak dlouho, vyjádřena v počtech bitů, se podílí bit ze vstupní nezabezpečené posloupnosti, na zabezpečovacím procesu, který se uskutečňuje v bloku kodéru nazvaném *Realizace zabezpečení.*

Nezabezpečený blok stromového kódu

$$u = m + 1 \,.\, u_0$$

Počet úseků nezabezpečené bitové posloupnosti, které jsou použity pro vytvoření zabezpečeného toku bitů.

Zabezpečený blok stromového kódu

$$v = (m + 1) \, . \, v_0$$

Princip funkčnosti dle schématu je následující. V pamětech (posuvném registru) $S_0, S_1$ kodéru jsou uloženy bity $\boxed{01}$ a bit který do kodéru vstupuje má hodnotu 1. Z tabulky 8.1 vidíme, že konečný stav v pamětech se změní na $\boxed{10}$ a hodnota na výstupu bude $\boxed{11}$. Můžeme to jednoduše dokázat.

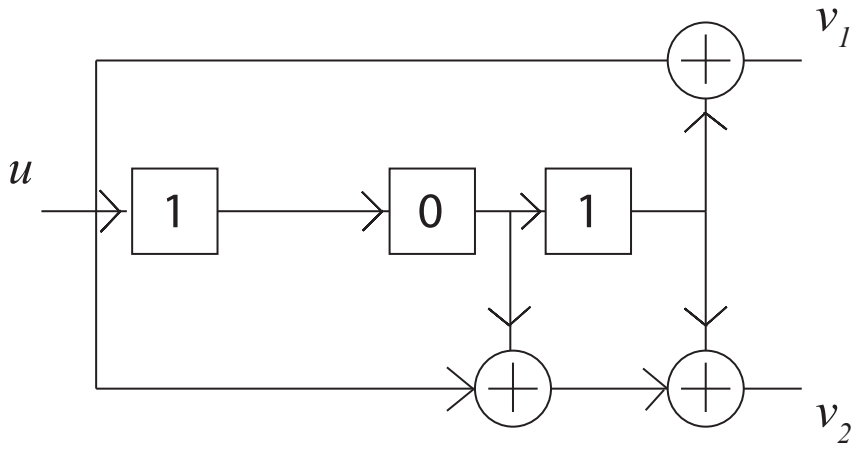
Vstupující bit $u_0 = 1$ se v horní větvi sečte ve sčítačče, provádějící operaci mod2 s druhým posuvným registrem a tím získame na výstupu $v_0 = 0$. V dolní větvy se $u_0$ sečte s hodnotou obou posuvných registrů,tedy na výstupu bude $v_1 = 0$ .

$$\begin{aligned} v_1 &= u_1 \oplus S_0 \\ v_1 &= 0 \\ v_2 &= u_1 \oplus S_0 \oplus S_1 \\ v_2 &= 0 \end{aligned}$$

Dále Vstupní bit $u_0$ se posune na na první místo paměti $S_0$ a následně se posune hodnota prvního registru $S_0$ do registru $S_1$.

Princip konvolučního kodéru se znázorňuje i pomocí stavového diagramu 8.2, mřížového nebo stromového grafu.

Stavový diagram: Informace obsažena v kruhu stavového diagramu znázorňuje konečný stav registru. Informace obsažena ve směru šipky znázorňuje vstupní bit a hodnotu výstupu.



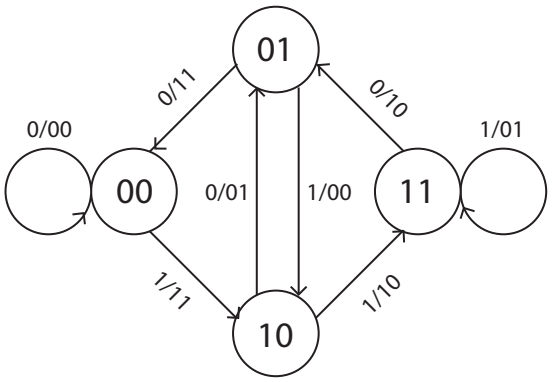

Obr. 8.1: Konvoluční kodér 1/2

Tab. 8.1: Vstupní bity, stavy, výstupní bity

| **Počáteční stav** $S_0[i]S_1[i]$ | **Informace** $u[i]$ | **Konečný stav** $S_0[i+1]S_1[i+1]$ | **Výstup** $v^0[i]v^1[i]$ |
|---|---|---|---|
| 00 | 0 | 00 | 00 |
| 00 | 1 | 10 | 11 |
| 01 | 0 | 00 | 11 |
| 01 | 1 | 10 | 00 |
| 10 | 0 | 01 | 01 |
| 10 | 1 | 11 | 10 |
| 11 | 0 | 01 | 10 |
| 11 | 1 | 11 | 01 |

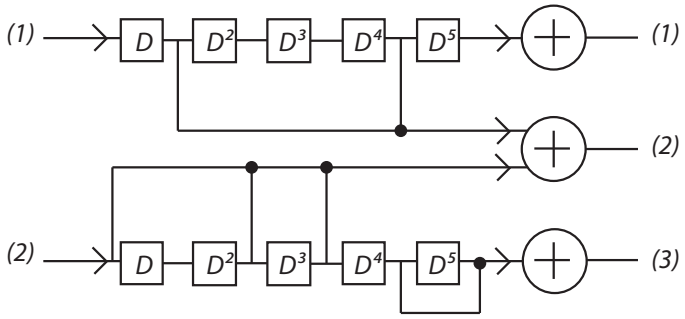
## 8.2. Kódování generujícími mnohočleny

Používáme operátor spoždění $S$ a vycházíme z předpokladu, že všechny signálové prvky mají stejnou časovou délku $S$. Dále máme generátor posloupnosti $G_j^i$, jehož horní index značí číslo vstupního dílčího toku a dolní číslo výstupního dílčího toku [5].




Obr. 8.2: Stavový diagram

Příklad: Nalezněte generující mnohočlen konvolučního kódu z následujícího obrázku 8.3:




Obr. 8.3: Konvoluční kód

$u_0 = 1, v_0 = 2, m = 5$ (spoždení o 5 jednotek), $v = 10, u = 12$ a $v = 18$.

## 8.3. Princip Dekódování

Dekódování se dělí na syndromové a postupné dekódování. Syndromové dekódování vychází z existence syndromu, jako výsledku kontroly správnosti přenesené posloupnosti bitů. Postupné dekódování využívá odchylek mezi přijatou posloupností bitů

a možnými posloupnostmi bitů, jak jsou například znázorněny T grafem vývoje kódování [5].

Pravděpodobnostní dekódování - je založeno na porovnání přijatého úseku zprávy s úseky zpráv ze seznamu užívaných zpráv. za vyslaný vybereme ten úsek zprávy, který se od přijatého úseku nejméně liší - je nejpravděpodobnější. Využíváme skutečnosti, že vývoj posloupnosti zabezpečené konvolučním kóden je znázornitelný některým z typů grafů, například mřížovým. Pro vyjadřování rozdílu mezi přijatou a vyslanou posloupností signálových prvků byla definována tzv. cena cesty. U dvojkových odpovídá minimální vzdálenosti Hammingova kódu $D_{min}$ 2.6, používané u blokových kódů. Při realizaci pravděpodobnostního dekódování se setkáváme se dvěma metodami:

1. Postupné dekódování - O správnosti přijaté posloupnosti se rozhoduje prvek po prvku.
2. Dekódování po úsecích - Viterbiho algoritmus. Úseky jsou porovnávány s užívanými úseky, což jsou části cest v mřížovém grafu. Za správnou se vybere ta cesta, která má nejmenší Hammingovu vzdálenost od přijatého úseku. Z ní se odvodí informační prvky. Tato metoda je známá jako Viterbiho dekódovací algoritmust. Tento algoritmus dekóduje přijatou posloupnost bitů po úsecích. Pro vysvětlení principu tohoto algoritmu se využívá možnosti znázornění vývoje zabezpečené posloupnosti mřížovým grafem.

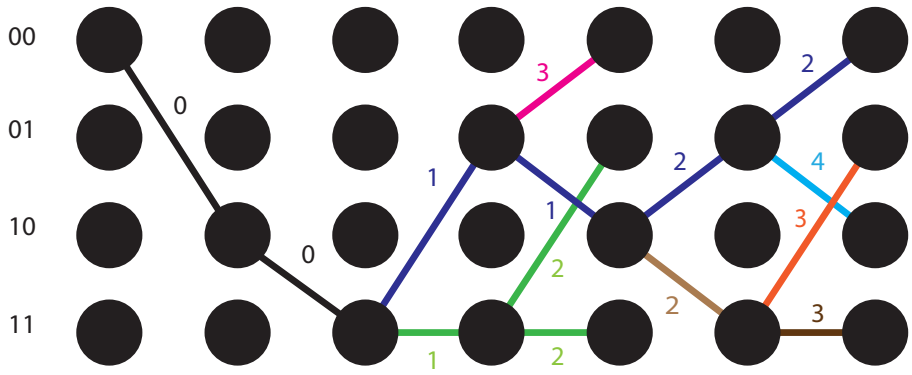
Příklad: Přenesený zabezpečený kód (2:1), obsahuje dvě chyby a je popsán mnohočlenem:

$$J(x) = \boxed{11\ 10\ 11\ 00\ 11\ 11}$$

K dekódování použijeme mřížový graf:

Řešení:

- Počátační stav v pamětech je $\boxed{00}$ . Z tabulky 8.1 víme, že výstup může být buďto $\boxed{00}$ nebo $\boxed{11}$ .
- Přijímač tedy porovná tento úsek podle velikosti Hammingovi vzdálenosti 2.4, s přijatým úsekem, který je $\boxed{11}$ . $D_{min}$ $\boxed{11}$ a $\boxed{11} = 0$ tudíž vybere tuto cestu.
- Momentálně je konečný stav v pamětech $\boxed{10}$ .
- Dále cesta vede ze stavu $\boxed{10}$ a možný konečný stav může být $\boxed{01}$ nebo $\boxed{10}$.
- Přijímač vybere výstup $\boxed{10}$, protože Hammingova vzdálenost = 0. Konečný stav je tedy $\boxed{11}$.




Obr. 8.4: Mřížový graf

- Počáteční stav v pamětech je 11 z tabulky víme, že výstup může být buďto 10 nebo 01. Očekávaný výstup tedy nekoresponduje ani s jedním očekávaným, graf bude pokračovat dvěmi směry. Metrika se zvýší o 1.
- Nejdříve se vydáme cestou 01. Výstup může být buďto 11 nebo 00. Výstup 00 odpovídá očekávané posloupnosti, naopak u výstupu 11 se zvýší metrika o 2.
- Vrátíme se ke stavu 11. Výstupy mohou být 10 a 01. U obou se metrika zvýší o 1 (očekávaná posloupnost je 00).
- Z grafu víme, že máme další 4 možné cesty kudy pokračovat. Dekodér by spočítal metriku pro všechny cesty, my však budeme pokračovat jen správnou cestou s metrikou 1.
- Počáteční stav v pamětech je 10 z tabulky víme, že výstup může být buďto 01 nebo 10. Výstup neodpovídá přijaté posloupnosti. U obou se zvýší metrika o 1. Abychom našli správnou cestu, budeme pokračovat oběma směry.
- Počáteční stav v pamětech je 11 z tabulky víme, že výstup může být buďto 10 nebo 01. Výstup neodpovídá přijaté posloupnosti. U obou se zvýší metrika o 1.
- Počáteční stav v pamětech je 01 z tabulky víme, že výstup může být buďto 11 nebo 00. 11 odpovídá přijaté posloupnosti, naopak u 00 se zvýší metrika o 2.
- Dekodér tedy opravil chyby a našel správnou cestu (nejmenší metrika). Metrika 2 znamená 2 chybé bity.

# 9 KONCEPT ŘEŠENÍ

Pro praktickou část bakalářské práce byly vytvořeny dva programy. První program názorně prezentuje probrané kódy, druhý slouží k experimentálním účelům uživatele.

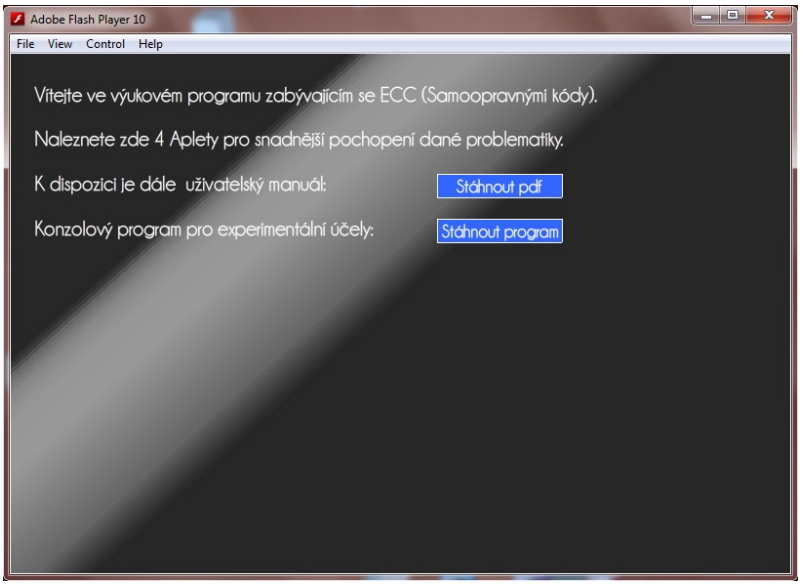
## 9.1. Apletový program

Tento program byl napsán v programu Adobe Flash Professional CS5.5 . Jedná se o grafický, vektorový program, který je vhodný pro tvorbu webových stránek, interaktivních animací, prezentací a her. Je kompatibilní s mnoha skriptovacími jazyky, konkrétně pro tento program byla využita nejnovější verze ActionScriptu 3.0. Všechny použité obrázky jsou vektorové a program tedy může být použit na jakémkoli zařízení.

ActionScript je objektově orientovaný programovací jazyk, který vychází z jazyku JavaScript. V našem případě zajišťuje správnou funkci celého funkčního bloku a všech tlačítek.

Výstup apletového programu můžeme spustit přes webový prohlížeč (.swf) nebo pomocí spouštěcího souboru (.exe). V případě webového prohlížeče je zapotřebí mít nainstalován Adobe Flash player 10.0 a novější.


Obr. 9.1: Apletový program - Webový prohlížeč Chrome




Obr. 9.2: Apletový program - spouštěcí soubor

### 9.1.1 Úvodní strana

Úvodní strana je tvořena z menu a prázdné stránky. Je naprogramována tak, aby po stisku jakéhokoli tlačítka v menu naběhl konkrétní aplet. K tomu slouží třída Loader v jazyku ActionScript 3.0. Vložený aplet se prezentuje jako dítě Loaderu.

Funkčnost je tedy následující: Po stisknutí tlačítka naběhne aplet s kterým chceme pracovat. Stiskneme-li v menu jiné tlačítko automaticky se odstraní právě spuštěný aplet a spustí se nový. (Vždy odstraníme dítě a nahradíme novým).

addEventListener je novinka v ActionScript 3.0. Pracuje se s ním následovně:

Je vytořena skupina tlačítek (objektů), které mají vlastní název instance. k objektu na který poukazujeme, přídáme addEventListener a do závorky vložíme událost která se má uskutečnit, a název funkce kterou má vykonat. Syntaxe je znázorněna na řádcích 24-28.

```
stop();
import flash.display.MovieClip;
import flash.display.Loader;
import flash.net.URLRequest;
var Xpos:Number = 0;
var Ypos:Number = 45;
var swf:MovieClip;
var loader:Loader = new Loader();
var context:LoaderContext = new LoaderContext();
context.applicationDomain = ApplicationDomain.currentDomain;
loader.x = Xpos;
loader.y = Ypos;
addChild(loader);

function btnClick(event:MouseEvent):void {
removeChild(loader);
var newSWFRequest:URLRequest = new URLRequest(event.target.name + ".swf");
loader.load(newSWFRequest);
loader.x = Xpos;
loader.y = Ypos;
addChild(loader);
}

aplet1.addEventListener(MouseEvent.CLICK, btnClick);
aplet2.addEventListener(MouseEvent.CLICK, btnClick);
aplet3.addEventListener(MouseEvent.CLICK, btnClick);
aplet4.addEventListener(MouseEvent.CLICK, btnClick);
aplet5.addEventListener(MouseEvent.CLICK, btnClick);
```

Obr. 9.3: AS3 Script pro menu

### 9.1.2 Aplet – Úvod

Aplet disponuje dvěma tlačitky (Stáhnout pdf, Stáhnout program). Po kliknutí na Stáhnout program se spustí stahování souboru. V případě tlačítka Stáhnout pdf se pdf otevře v prohlížečí, z kterého je možné jej stáhnout. V našem případě je program i pdf uloženo v lokálním adresáři. Stahování probíhá pomocí funkce navigateToURL, která otevírá po kliknutí tzv. vyskakovací okna.

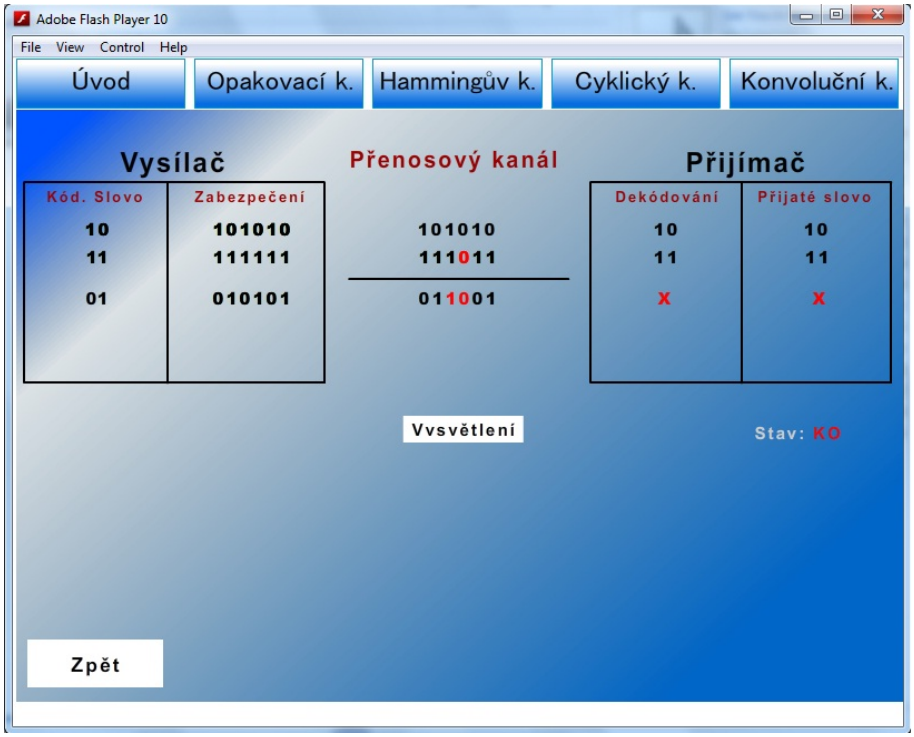
```
stop();

pdf.addEventListener(MouseEvent.CLICK, linkToResume, false, 0, true);

function linkToResume(evt:MouseEvent):void {
     navigateToURL(new URLRequest("xpetri16.pdf"));
}

exe.addEventListener(MouseEvent.CLICK, linkToResume2, false, 0, true);

function linkToResume2(evt:MouseEvent):void {
     navigateToURL(new URLRequest("program.exe"));
}
```

Obr. 9.4: AS3 Script úvodní stránky

### 9.1.3 Aplet – Opakovací kód

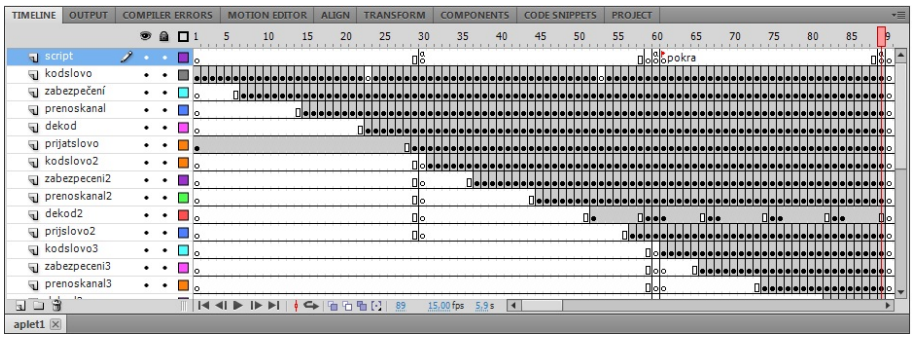
Aplet je koncipován jako kódové slovo přenášené od zdroje k přijímači. Znázorňuje celkem tři situace.

- V přenosu nenastala žádná chyba, přenos proběhl úspěšně. Po kliknutí na Vysvětlení se uživatel dozví jak se kódové slovo tvoří a jak se zabezpečuje.
- V přenosu nastala jedna chyba, přijímač chybu opraví. Po kliknutí na Vysvětlení se uživatel dozví jakým způsem se chyba opraví.
- V přenosu nastaly dvě chyby, přijímač nedokáže chybu opravit. Po kliknutí na vysvětlení se uživatel dozví, že byla překročena korekční schopnost.




Obr. 9.5: AS3 Script úvodní stránky

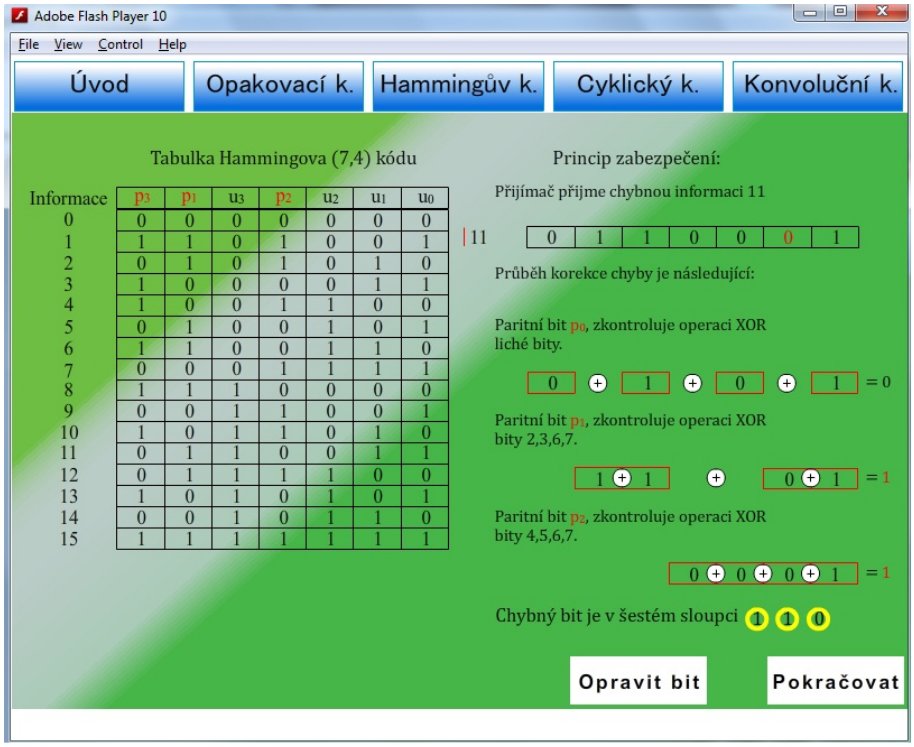
V programovacím jazyce ActionScript 3.0 jsou programovány všechna tlačítka, způsobem, že se odkazují na určité úseky ve vytvořeném apletu, aby uživatel mohl intuitivně ovládat program. V Adobe Flash Professional CS5.5 byly vytvořeny animace za pomoci tzv. Motion Tween. Ten funguje tak, že vytvoříme libovolně dlouhý časový úsek ve kterém se bude objekt pohybovat.




Obr. 9.6: Hammingův kód – Adobe Flash Professional

### 9.1.4 Aplet – Hammingův kód

Aplet začíná Hammingovým (7,4) kódem a názornou ukázkou, na jakých pozicích pracují kontrolní bity. Dále je znázorněna chyba v přenosu, kde je zaměněn bit v šestém míste kódového slova. Je vysvětlen průběh korekce chyby i s nalezením pozice chybného bitu. Dále si uživatel může bit opravit a pokračovat k Hammingovu (8,4) kódu. Zde je vysvětlen princip přidaného paritního bitu na velmi podobném příkladu s jednou chybou. Dále jsou naznačeny postupy pro další situace které mohou nastat.




Obr. 9.7: Vytváření snímků

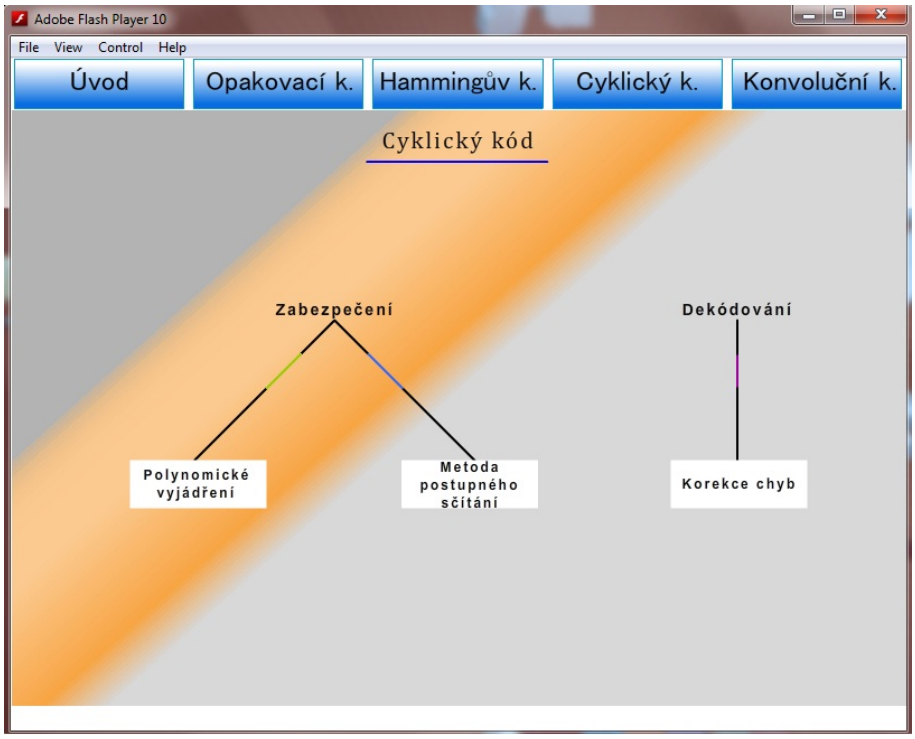
### 9.1.5 Aplet – Cyklický kód

Cyklický kód se v úvodním menu rozděluje na tři části. První dvě části jsou o cyklickém zabezpečení, třetí část je věnována dekódování.

1) Polynomické vyjádření – Graficky a velmi důkladně popisuje průběh zabezpečení cyklickým kódem (15,7) metodou polynomického vyjádření. Je tvořen ze statických obrazů a krátkých animací, kde pracují dvě animace současně. První animace znázorňuje výsledek dělení. Druhá animace znázorňuje způsob jak se k výsledku uživatel dopracuje.

2) Metoda postupného sčítání – Graficky a velmi důkladně popisuje průběh zabezpečení cyklickým kódem (15,7) metodou postupného sčítání. Povětšinou je tvořen ze statických obrazů. Ty jsou vždy do dalšího snímku zkopírovány a navíc se doplní o další informaci. Výsledkem je animace, která tvoří dojem, že program počítá postupně, stejně jako uživatel.

3) Korekce chyb – Metoda postupného sčítání – Graficky a velmi důkladně popisuje průběh dekódování cyklického kódu (15,7) a následně jeho korekci. V animaci jsou použity stejné principy jako v předešlých dvou animacích.




Obr. 9.8: Úvodní menu Cyklického kódu

Z důvodu velkého počtu snímků, byl změněn systém tlačítek. Uživatel má všechny stránky očíslovány v pravém dolním rohu. Stačí kliknout na konkrétní číslo stránky a dojde k načtení obsahu stránky. Mezi stránkami se může uživatel libovolně pohybovat. Číslo konkrétní stránky je vždy zvýrazněno. Na poslední straně je možnost vrátit se do menu. K vytvoření takové navigace se musela všechna tlačítka vložit do jednoho pole a vytvořit funkci, která na klik myši zobrazí příslušnou stránku.


Obr. 9.9: Číslování stránek

### 9.1.6 Aplet – Konvoluční kód

Konvoluční kód se v úvodním menu dělí na Konvoluční kodér, Generující mnohočleny a Mřížový graf.

1) Konvoluční kodér – K vysvětlení principu je použito schéma konvolučního kodéru a tabulky do které se postupně doplňují informace. Ke zjištění výstupu je použita animace, kde jsou barevně vyznačeny cesty s vysvětlením. Ke zjištění koncového stavu je použita animace, kde se aktuální informace na vstupu posouvá na místo $S_0$ v registru a informace obsažená v $S_0$ na místo $S_1$.

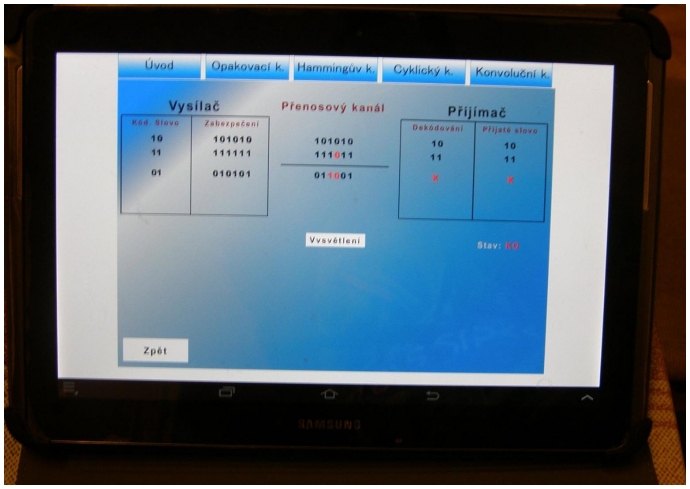
2) Generující mnohočleny – Bylo vytvořeno jednoduché schéma na kterém je pomocí statických obrázků vysvětlen princip činnosti.

3) Mřížový graf – V této prezentaci je podrobný teoretický úvod na který navazuje schéma mřížového grafu. Jak se přijatý mnohočlen postupně dekóduje, v mřížovém grafu přibývají cesty. Pro snazší pochopení se prezentace doplnila o tabulku, která byla použita u konvolučního kodéru.


Obr. 9.10: Úvodní menu Konvolučního kódu

### 9.1.7 Aplet pro operační systém Android

Operační systém android využívá k instalování aplikací koncovku .apk. V případě apletu, který byl vytvořen stačí všechny vytvořené soubory překopírovat do nového projektu v Adobe AIR pro android. Systém trochu poupravit a publikovat jako Android aplikaci. Výsledkem je další rozšíření vytvořených aplikací k dalším uživatelům využívající jiný operační systém.




Obr. 9.11: Aplet pro operační systém Android

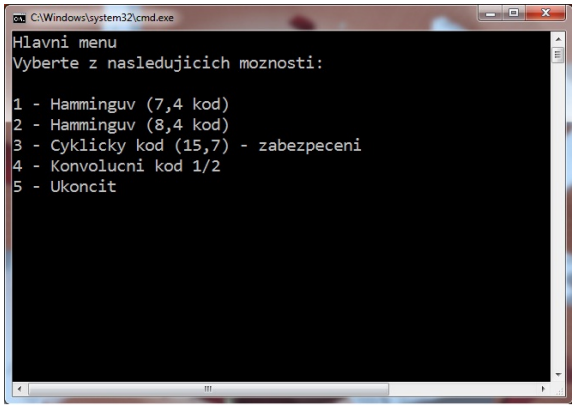
## 9.2. Experimentální program

Program byl napsán jako konzolová aplikace programovacího jazyka C++. Konzolová aplikace přijímá vstupy a posílá je na výstup konzole, která se také nazýva příkazový řádek. Může komunikovat s dalšími desktopovými aplikacemi skrze roury (pipes) nebo přes jiné RPC mechanismy. Program má příponu .exe a je spouštěn v příkazovém řádku.

Zadávání uživatelských vstupů - Vždy se zadá jeden vstup (0 nebo 1), následně uživatel stiskne klávesu enter a zadá další vstup.

### 9.2.1 Úvodní strana

Úvodní stranu tvoří menu, kde má uživatel možnost vybrat si ze čtyř probíraných kódů. Menu bylo naprogramováno pomocí switch - case.

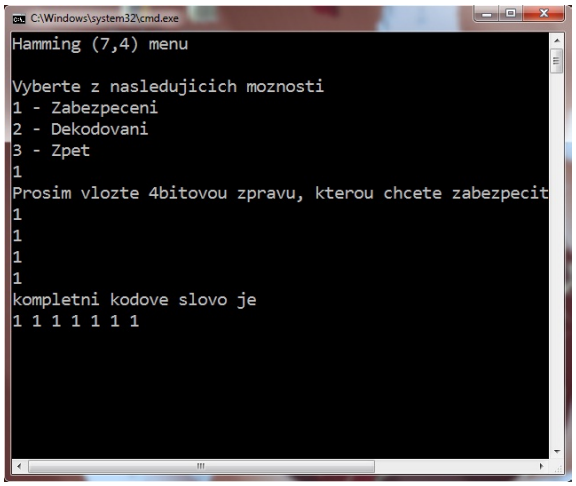



Obr. 9.12: Úvodní menu programu

### 9.2.2 Program - Hammingův (7,4) kód

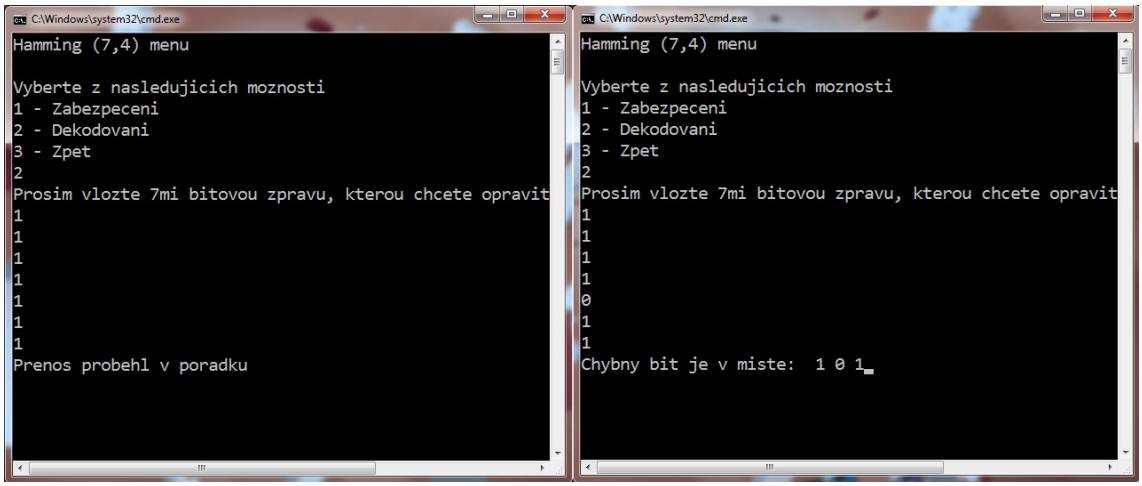
Program je dělen na zabezpečení a dekódovaní.

Při zabezpečení vloží uživatel čtyři informační bity, který mu následně program zabezpečí.




Obr. 9.13: Zabezpečení Hammingova (7,4) kódu

Při dekódování vloží uživatel kódové slovo o délce sedmi bitů , které program dekóduje a najde místo chyby. Přípustná je maximálně jedna chyba.




Obr. 9.14: Dekódování Hammingova (7,4) kódu

Kód v C++: Nejdříve se použil cyklus for a jednotlivé vstupy se ukládají do $array[i]$. Dále se spočítali paritní bity $array2[0]$, $array2[1]$ a $array2[2]$, které se uložili do druhého pole. Třetí pole $array3$ slouží k uchování správného sledu bitů. Princip s poli je u dekódování obdobný. Navíc byla použita podmínka if–else z důvodu, že může nastat buďto bezchybný přenos, nebo chybný, který poukáže na chybné místo v kódovém slově.

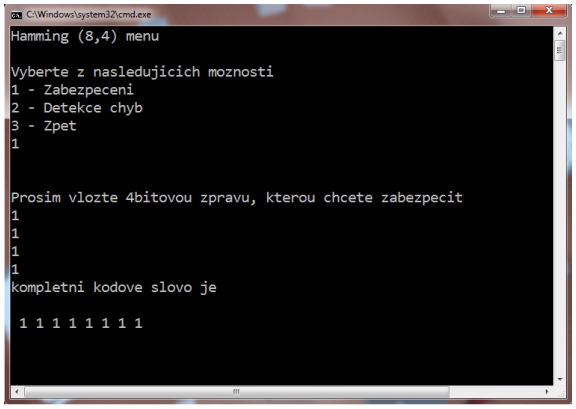
```
                case 1:


cout << "Prosim vlozte 4bitovou zpravu, kterou chcete zabezpecit \n";
for(int i = 0; i < size; i++) {
    cin >> array[i];
}
getchar;
array2[0] = (array[0]^array[1]^array[3]);
array2[1] = (array[0]^array[2]^array[3]);
array2[2] = (array[1]^array[2]^array[3]);
printf("kompletni kodove slovo je \n");
array3[0] = array2[0];
array3[1] = array2[1];
array3[2] = array[0];
array3[3] = array2[2];
array3[4] = array[1];
array3[5] = array[2];
array3[6] = array[3];
cout << array3[0] << " " << array3[1] << " " << array3[2] << " " << array3[3] << " " << array3[4] << " " << array3[5]
<< " " << array3[6];
getchar();
getchar();
                break;
                case 2:
cout << "Prosim vlozte 7mi bitovou zpravu, kterou chcete opravit (max 1 chyba) \n";
for(int i = 0; i < celkove; i++) {
cin >> array3[i];
}
getchar;
array2[0] = (array3[0]^array3[2]^array3[4]^array3[6]);
array2[1] = (array3[1]^array3[2]^array3[5]^array3[6]);
array2[2] = (array3[3]^array3[4]^array3[5]^array3[6]);
if (array2[2] == 0 && array2[1] == 0 && array2[0] == 0) {
cout << "Prenos probehl v poradku";
}
else {
cout << "Chybny bit je v miste: " << " " << array2[2] << " " << array2[1] << " " << array2[0];
}
getchar();
```

Obr. 9.15: Hammingův (7,4) kód v C++

### 9.2.3 Program - Hammingův (8,4) kód

Program je dělen na zabezpečení a dekódovaní.

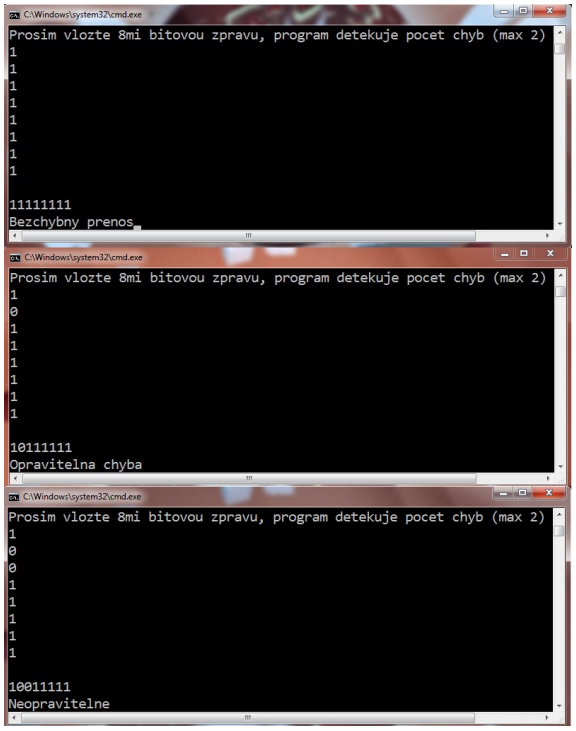
Stejně jako u předchozího kódu vloží uživatel čtyři informační bity, který mu následně program zabezpečí.




Obr. 9.16: Zabezpečení Hammingova (8,4) kódu

Při dekódování vloží uživatel kódové slovo o délce osmi bitů , které program dekóduje

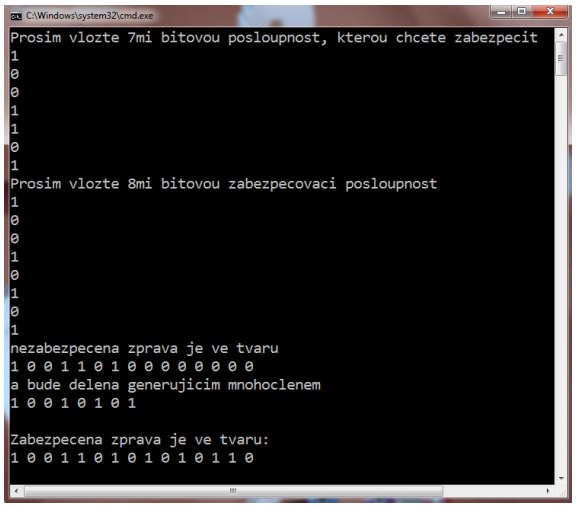
a zjistí zda proběhl přenos v pořádku, nebo jestli je kódová posloupnost opravitelná. Třetí možností je, že program detekuje dvě chyby a v tom případě kódovou posloupnost nelze opravit.




Obr. 9.17: Dekódování Hammingova (8,4) kódu

### 9.2.4 Program - Cyklický (15,7) kód

Program byl nastaven tak, že uživatel zadá posloupnost kterou chce zabezpečit a dále vloží osmi bitovou zabezpečovací posloupnost. Následně se vypíše výsledek.




Obr. 9.18: Zabezpečení Cyklického (15,7) kódu

Kód v C++:

```
 cout << "Prosim vlozte 7mi bitovou posloupnost, kterou chcete zabezpecit \n";
     for(int i = 0; i < informacni; i++) {
    cin >> pole[i];}
getchar;
cout << "Prosim vlozte 8mi bitovou zabezpecovaci posloupnost \n";
for(int i = 0; i < paritni; i++) {
cin >> pole2[i];
}
pole3[0] = pole[0]; pole3[1] = pole[1];  pole3[2] = pole[2];  pole3[3] = pole[3];  pole3[4] = pole[4];  pole3[5] = pole[5];
pole3[6] = pole[6];  pole3[7] = 0;  pole3[8] = 0;  pole3[9] = 0;  pole3[10] = 0;  pole3[11] = 0; pole3[12] = 0;
pole3[13] = 0; pole3[14] = 0;
cout << "nezabezpecena zprava je ve tvaru \n" << pole3[0] << " " << pole3[1] << " " << pole3[2] << " " << pole3[3] << " "
<< pole3[4] << " " << pole3[5]
<< " " << pole3[6] << " " << pole3[7] << " " << pole3[8] << " " << pole3[9] << " " << pole3[10] << " " << pole3[11]
<< " " << pole3[12] << " " << pole3[13] << " " << pole3[14] << "\n";
getchar();
cout << "a bude delena generujicim mnohoclenem \n"  << pole2[0] << " " << pole2[1] << " " << pole2[2] << " " << pole2[3]
<< pole2[4] << " " << pole2[5] << " " << pole2[6] << " " << pole2[7] << " \n" ;
getchar();
for (int i = 0; i < 15; i++){
pole4[i] = pole3[i]; }
for (int i = 0; i < 15 - 7; i++)
{
    if (pole4[i] == 1)
    {

        for (int j = 0; j < 8; j++)
        {
            pole4[i+j] = pole4[i + j] ^ pole2[j];
        }
    }
}
for (int i = 0; i < 8; i++)
{
    pole4[i] = pole3[i];
}
cout << "Zabezpecena zprava je ve tvaru: \n";
    for (int i = 0; i < 15; i++)
{
    cout << pole4[i] << " ";
}
```

Obr. 9.19: Cyklický (15,7) kód v C++

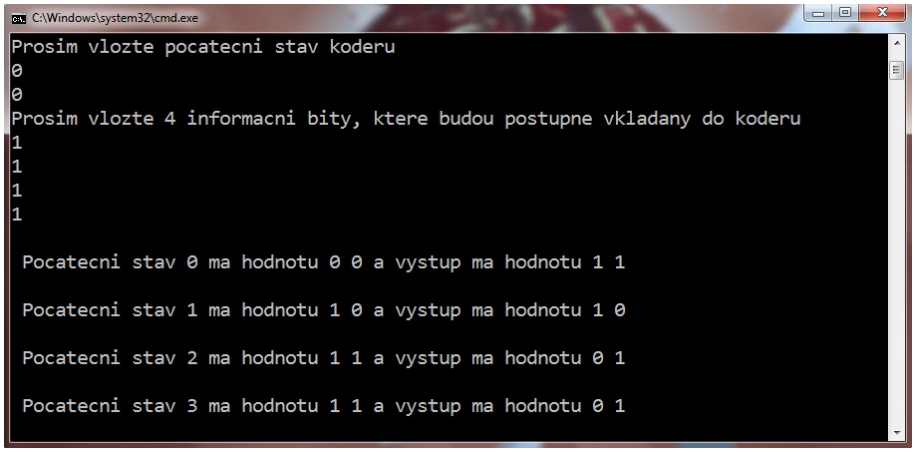
Program v C++: V prvním cyklu se prochází (15-7) prvků a kontroluje zda je daný prvek roven jedné. Pokud podmínka platí, tak se v následujícím cyklu projde dalších (15-7) prvků, počínaje první nalezenou jedničkou. Na každém z těchto prvků se provede operace XOR s prvkem druhého pole (Generující mnohočlen). Proces skončí v době, kdy již nebude nalezena žádná jednička v osmi prvkovém poli.

### 9.2.5 Program - Konvoluční kód

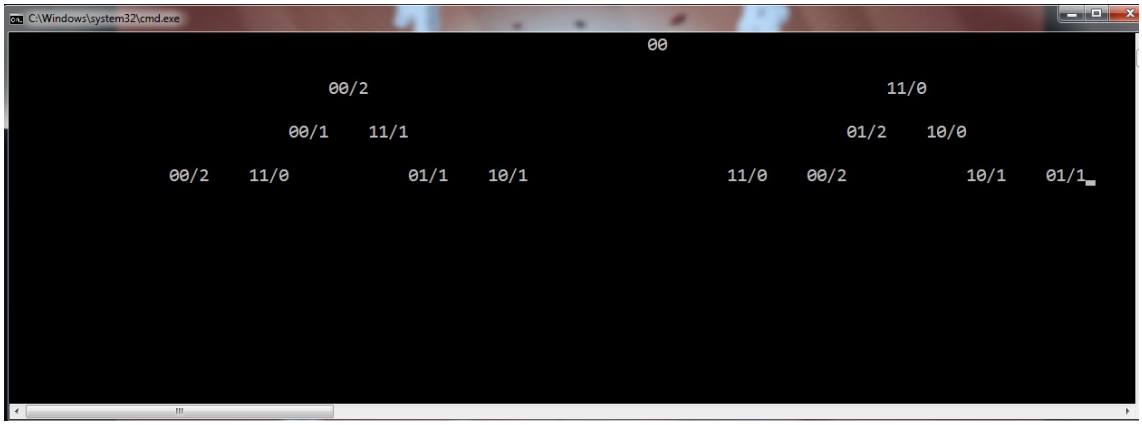
Program je dělen na zabezpečení a dekódovaní.

Zabezpečení: Uživatel zadá počáteční stav kodéru 8.1 a čtyři informační bity které do něho budou postupně vkládány. Program vypíše počáteční stavy a výstupy kodéru po každém bitu.

Dekódování: Uživatel zadá počáteční stav a šesti bitovou posloupnost výstupů, které chce dekódovat. Program vypíše možné výstupy a vzdálenost metrik do stromového grafu, pro lepší přehlednost. Pokud se graf nezobrazí správně, nastavte ve vlastnostech příkazového řádku Screen Buffer Size na 400 a program spusťte znovu.




Obr. 9.20: Zabezpečení Konvolučního kódu




Obr. 9.21: Dekódování Konvolučního kódu

Program v C++: Při zabezpečení se po zadaní všech vstupů vypočítájí výstupy, tyto dvě informace program vypíše a uloží si do vektoru nové počáteční stavy, ke kterým

se spočítá výstup. Při dekódování uživatel může zadat šesti bitový výstup který mu vyšel a počáteční stav při kterém kód řešil. Byla použita matice, do které byly na každý řádek zapsány počáteční stavy kodéru, konečné stavy kodéru a jejich výstupy. Cyklus for vyhledá řádek, který se rovná zadaným hodnotám počátečního stavu uživatele. Ten se uloží do pomocné proměnné *pom*. Víme, že existují dvě možnosti a ta druhá je vždy o jeden řádek v matici níž. Proto uložíme i druhý řádek do pomocné proměnné *pom*2. Dále spočítáme metriku porovnáním zadaných výstupů uživatelem od výstupů zadaných v matici. K tomu nám posloužila podmínka if-else. Nakonec uložíme nové počáteční stavy.

```
cout << "Prosim vlozte pocatecni stav koderu \n";
for(int i = 0; i < stav; i++) {
    cin >> array001[i];}
getchar;
cout << "Prosim vlozte 4 informacni bity, ktere budou postupne vkladany do koderu\n";
for(int i = 0; i < vstup; i++) {
    cin >> array000[i];}
getchar;
for (int i=0; i<vstup; i++) {
    array002[0] = (array000[i] ^ array001[1]);
    array002[1] = (array000[i] ^ array001[0] ^ array001[1]);
    cout <<"\n"<< " Pocatecni stav" << " " << i << " " << "ma hodnotu" << " " << array001[0] << " " << array001[1]
    << " " <<  "a" << " " << "vystup" << " "  << "ma hodnotu" << " " << array002[0] << " " << array002[1] << "\n";
    array001[1] = array001[0];
    array001[0] = array000[i];
```

Obr. 9.22: Zabezpečení Konvolučního kódu v C++

```
    cout << "zadejte vstupni slovo\n";
    for (i=0;i<6;i++){
        cin >> vek[i];
    }

    cout << "zadejte pocatecni stav\n";
    for (i=0;i<2;i++){
        cin >> poc[i];
    }

    for (i=0;i<8;i++){
    if (poc[0] == matice [i][0]&&poc[1] == matice[i][1]) {
        pom=i;
        pom2=pom+1;
        i=7;
    }}
    system("cls");
    cout << "                                                                    " <<poc[0] << poc[1] << "\n\n";
if (vek[0]!=matice[pom][4]){shoda=shoda+1;}
    if (vek[1]!=matice[pom][5]){shoda=shoda+1;}
    if (vek[0]!=matice[pom2][4]){shoda2=shoda2+1;}
    if (vek[1]!=matice[pom2][5]){shoda2=shoda2+1;}
    vektro[0][0] = matice[pom][4];
    vektro[0][1] = matice[pom][5];
    vektro[1][0] = matice[pom2][4];
    vektro[1][1] = matice[pom2][5];
    poc[0]=matice[pom][2];
    poc[1]=matice[pom][3];
    poc2[0]=matice[pom2][2];
    poc2[1]=matice[pom2][3];
    cout << "                              " << vektro[0][0] << vektro[0][1] << "/" << shoda << "
```

Obr. 9.23: Ukázka dekódování Konvolučního kódu v C++

# 10 ZÁVĚR

Cílem této práce bylo nastudovat problematiku korekčních kódů a na tomto základě navrhnout koncept řešení.

Kódy byly zvoleny tak, aby čtenář postupoval od lépe pochopitelných kódů na kterých by měl pochopit základní principy po kódy složitější. Na začátek jsem zvolil jednu sekci důležitých pojmů, bez kterých se čtenář těžko orientuje. Další sekce už se týkaly korekčních kódu, které byly stručně popsány a hlavním záměrem bylo ukázat a vysvětlit princip jakým způsobem se dají zabezpečit popřípadě dekódovat. V této práci jsem psal o šesti kódech, začínaje blokovými a konvolučními konče. Dle mého názoru má každý z nich v této práci své místo a rozvíjí o další poznatky.

V praktické části byl nejdříve vytvořen interaktivní Aplet, který je spjatý s teoretickou částí a dává uživatelům možnost si probírané učivo detailně osvojit. Je multiplatformní, tedy dá se využít jak na tabletech nebo telefonech se systémem Android, tak pro počítače se systémem Windows a i jako webová stránka pro všechna zařízení mající přístup k internetu. Poté následovalo vytvoření experimentálního programu, kde si uživatel může vyzkoušet, zda látku pochopil.

Práce je psána pro studenty, kteří mají zájem o problematiku korekčních kódů. Tato práce by je mohla oslovit a usnadnit jim tak jejich první kroky. Úsilí bylo kladeno na interaktivitu a názornost, která vede k lepšímu porozumnění a to se snad podařilo splnit.

# LITERATURA


[1] BARRY, J. R.; Lee, E. L.; Messerschmitt, D. G. *Digital communication*, 3. vyd. Springer, 2003, 838 s., ISBN 0–7923–7548–3

[2] FROLKA, J. *BCH kódy*, Diplomová práce. Brno: Vysoké učení technické v Brně, Ústav telekomunikací, 2012. 58 s.

[3] MORELOS-ZARAGOZA, R. H. *The Art of Error Correcting Coding* , 2. vyd., 2006, 278 s., ISBN 978-0-470-01558-2

[4] RITTER, T. *The Great CRC Mystery* , Dr. Dobb's Journal 11 (2), 2009

[5] ŠILHAVÝ, P. *Datová komunikace*, Brno: FEKT Vysokého Učení Technického v Brně, 2012, 211 s., ISBN 978–80–214–4455–3

[6] WARREN, H. S. *Hacker's Delight*, 2. vyd. Addison-Wesley Professional, 2013, 396 s., ISBN 0–321–84268–5