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

TURBO KONVOLUČNÍ A TURBO BLOKOVÉ KÓDY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

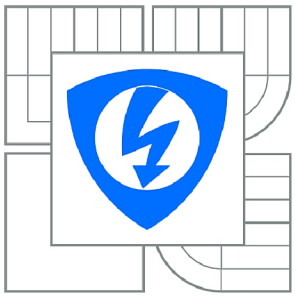
AUTOR PRÁCE
AUTHOR

Bc. JAKUB ŠEDÝ

BRNO 2011

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

TURBO KONVOLUČNÍ A TURBO BLOKOVÉ KÓDY

TURBO-CONVOLUTION AND TURBO-BLOCK CODES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

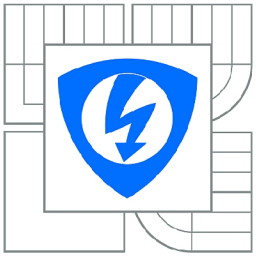
AUTOR PRÁCE — Bc. JAKUB ŠEDÝ

AUTHOR

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

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Jakub Šedý	***ID:***	98460
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Turbo konvoluční a turbo blokové kódy

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku protichybového zabezpečení pomocí turbo konvolučních a turbo blokových kódů. Dále se zaměřte na problematiku dekódování zprávy zabezpečené uvedenými kódy a porovnejte jednotlivé přístupy. V programovém prostředí Matlab vytvořte demonstrační program, který bude možno využít jako výukovou pomůcku a rovněž jako analyzačního nástroje pro porovnání dosažených parametrů kódů. Program bude zahrnovat grafické rozhranní a jednotlivé položky v něm doplňte nápovědou. S pomocí vytvořeného programu realizujte zevrubné srovnání dosažených parametrů (kódového zisku a dalších) pro různé parametry kódů.

DOPORUČENÁ LITERATURA:

[1] Hanzo, L., Liew, T. H., Yeap, B. L., Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels. JohnWiley, 2002, ISBN: 0470847263.
[2] Moon, T. K. Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, 2005, ISBN-13: 978-0070010697.
[3] Lin, S., Costello, D. J.. Error Control Coding: Fundamentals and Applications, second edition, Prentice Hall: Englewood Cliffs, NJ, 2005 ISBN: 0-13-042672-5.
[4] Lin, S., Costello, D. J.. Channel Coding in Communication Networks: From Theory to Turbocodes, Wiley-ISTE, 2007 ISBN: 978-1-90520-924-8.

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

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

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

ABSTRAKT

Cílem práce je přiblížit čtenáři problematiku turbo konvolučních a turbo blokových kódů a to v oblasti dekódování zabezpečené zprávy. Praktická část je zaměřena na návrh demonstračního programu v programovém prostředí Matlab. Práce je členěna do čtyř základních částí. První dvě se zabývají teoretickým rozborem kódování a dekódování. Třetí část obsahuje popis vytvořeného demonstračního programu, který umožňuje procházet proces kódování a dekódování. Čtvrtá je věnována simulacím a výkonnosti turbo kódů.

KLÍČOVÁ SLOVA

Turbo konvoluční kódy, turbo blokové kódy, kodér, dekodér, SOVA, RSC, BCH, LLR, Matlab.

ABSTRACT

The aim is to explain the Turbo convolutional and block turbo codes and decoding the secure message. The practical part focuses on the design of a demonstration program in Matlab. The work is divided into four parts. The first two deal with theoretical analysis of coding and decoding. The third section contains a description created a demonstration program that allows you to navigate the process of encoding and decoding. The fourth is devoted to simulation and performance of turbo codes.

KEYWORDS

Turbo convolutional codes, turbo block codes, encoder, decoder, SOVA, RSC, BCH, LLR, Matlab.

ŠEDÝ, Jakub *Turbo konvoluční a turbo blokové kódy*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 85 s. Vedoucí práce byl Ing. Pavel Šilhavý, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Turbo konvoluční a turbo blokové kódy“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Turbo kódy byly navrženy v roce 1993. Na návrhu se nejvíce podíleli Claude Berrou, Alain Glavieux a Punya Thitimajshima. Základními prvky těchto kódů jsou konvoluční kódy a Viterbiho algoritmus, ale jak později ukázal Hagenauer, je možné použít i kódy blokové. Tyto kódy vykazují velký kódový zisk i při vysokých datových rychlostech, a to při relativně jednoduché realizaci. Svými vlastnostmi jsou schopny zajistit nízkou bitovou chybovost BER, při malých hodnotách poměru signál/šum, který se blíží Shannonovu limitu.

Cílem práce, jak je uvedeno v zadání, je nastudovat problematiku protichybového zabezpečení pomocí turbo konvolučních a turbo blokových kódu. Dále se zaměřit na problematiku dekódování zprávy zabezpečené uvedenými kódy. V programovém prostředí Matlab vytvořit demonstrační program, který bude možno využít jako výukovou pomůcku a rovněž jako analyzačního nástroje pro porovnání dosažených parametrů kódu. Program bude zahrnovat grafické rozhraní. S pomocí vytvořeného programu realizovat zevrubné srovnání dosažených parametrů pro různé parametry kódu. V první části je teoreticky rozebrána problematika turbo konvolučního kódování a dekódování. Turbo blokové kódy, jejich kódování a dekódování je řešeno v druhé části. Následující třetí a čtvrtá kapitola je věnována samotnému návrhu programu v prostředí Matlab, simulacím a srovnání dosažených výsledků.

1 TURBO KONVOLUČNÍ KÓDY

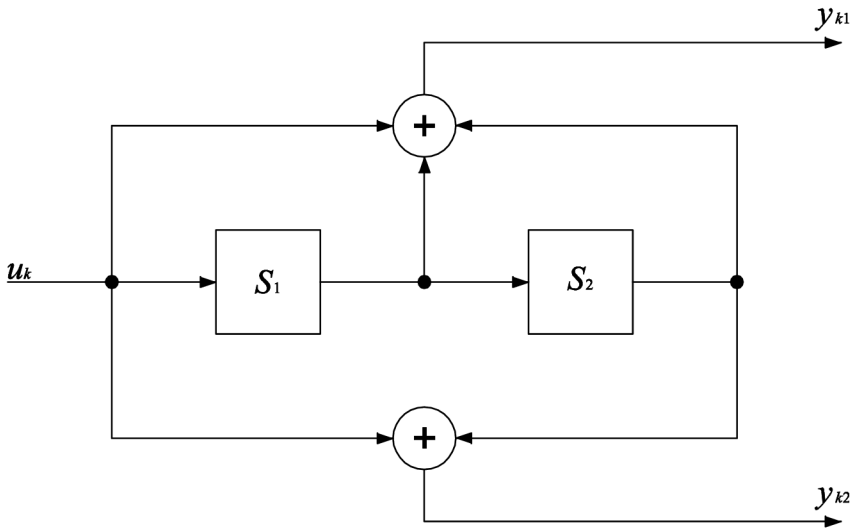
1.1 Úvod

V této kapitole si ukážeme, jakým způsobem pracuje turbo kódování, které využívá konvoluční kódy. Vstupní posloupnost bitů je kódována dvěma kodéry, mezi kterými je uložen prokladač [6], který zajistí, že kódované posloupnosti budou na sobě vzájemně nezávislé. Často se používají RSC (rekurzivní systematický konvoluční – Recursive Systematic Convolutional) kodéry [5] [6], kde každý RSC kodér produkuje systematický výstup, který je ekvivalentní se vstupní informací a dále produkuje paritní bity. Obě paritní posloupnosti je možné zúžit (děrovat) [8] před tím, než budou i se systematickými bity přeneseny do dekodéru. Pomocí zúžení je možné snížit počet paritních bitů na polovinu, a tím také snížit informační rychlost na 1/2.

Pro dekódování musí být použity speciální algoritmy, které používají tzv. měkký vstup a měkký výstup [3] [5] [7] [10]. Tyto měkké vstupy a výstupy neurčují pouze, jestli dekódovaný bit má hodnotu 0 nebo 1, ale také vypočítávají pravděpodobnost, jestli byl bit správně dekódován. Turbo dekodér pracuje iterativně. První iterace prvního dekodéru dává odhad původní sekvence dat, založené na měkkém vstupu kanálu. Poskytuje také vnější výstup. Vnější výstup pro dané bity není založený na vstupním kanále pro tento bit, ale na zprávě pro okolní bity a omezení vyplývajících z použitého kódu. Tento vnější výstup z prvního dekodéru se použije jako a-priori informace pro druhý dekodér zároveň se vstupní informací. Druhý dekodér nám dá opět vnější informaci a měkký výstup. Ve druhé iteraci se vnější informace z druhého dekodéru použije jako a-priori informace pro kodér první, a tím může dekódovat více správných bitů, než tomu bylo v první iteraci. Tento cyklus se stále opakuje a v každé iteraci obou dekodérů se vypočítá měkký výstup a vnější informace na základě vstupní posloupnosti a a-priori informace získané z vnějších hodnot předchozího dekodéru. Po každé iteraci se BER (bitová chybovost – Bit Error Rate) snižuje. Obvykle se používá 4 až 14 iterací.

1.2 Konvoluční kódování

RSC (n, k, m) kodér je odvozen z NRC (nerekurzivní konvoluční – Non-Recursive Convolutional) kodéru [6] obr. 1.1. Jeden z jeho výstupů tvoří zpětnou vazbu přivedenou na vstup kodéru.



Obr. 1.1: NRC (2, 1, 2) kodér.

Základním prvkem těchto kodérů jsou posuvné registry, které uchovávají hodnoty příchozích bitů a logické členy XOR. Výstupní posloupnost je dána kombinací současných a předchozích vstupních hodnot. Vytvářecí mnohočleny pro nerekurzivní nesystematický kodér jsou:

$$\begin{aligned} \mathbf{G}_{(1)}^{(1)} &= 1 + D + D^2, \\ \mathbf{G}_{(2)}^{(1)} &= 1 + D^2. \end{aligned}$$

Vytvářecí mnohočlen pro RSC kodér je možné odvodit z mnohočlenů NRC kodéru a je dán jako [6] [5]:

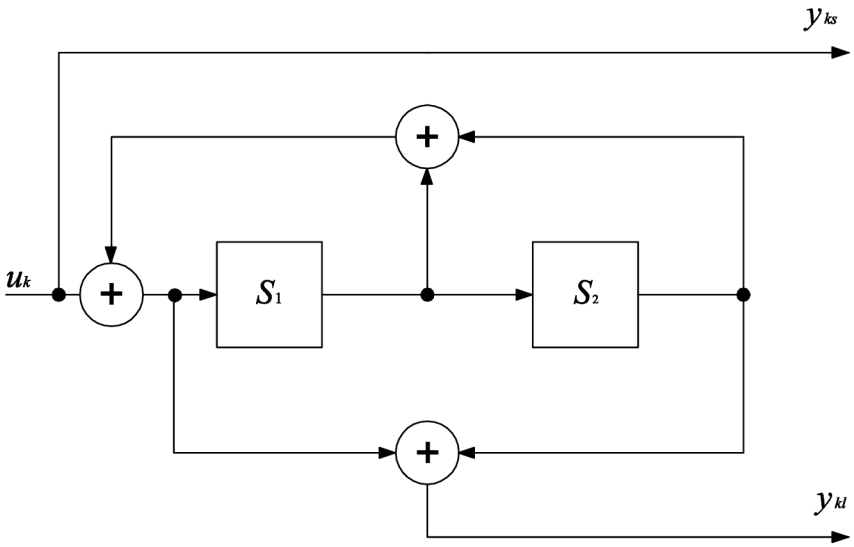
$$\mathbf{G}(D) = \left[\mathbf{G}_{(1)}^{(1)}, \mathbf{G}_{(2)}^{(1)}\right], \tag{1.1}$$

$\mathbf{G}_{(1)}^{(1)}$ značí systematický výstup kodéru a $\mathbf{G}_{(2)}^{(1)}$ lze zapsat jako:

$$\mathbf{G}_{(2)}^{(1)} = \frac{\mathbf{G}_p}{\mathbf{G}_z}, \tag{1.2}$$

kde $\mathbf{G}_p$ reprezentuje dopředný výstup a $\mathbf{G}_z$ zpětnou vazbu do vstupu kodéru. Vytvářecí mnohočleny pro RSC kodér obr. 1.2 jsou:

$$\begin{aligned} \mathbf{G}_{(1)}^{(1)} &= 1, \\ \mathbf{G}_{(2)}^{(1)} &= \frac{1+D^2}{1+D+D^2}. \end{aligned}$$



Obr. 1.2: RSC (2, 1, 2) kodér.

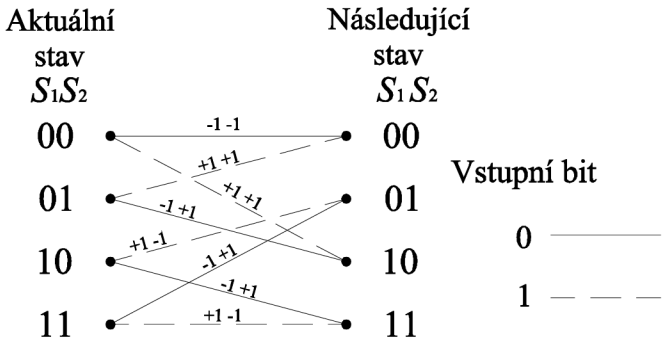
Základními parametry jsou informační rychlost R a délka kódového ohraničení K. Informační rychlost udává poměr počtu bitů vstupujících a vystupujících z kodéru

$$R = \frac{k}{n}. \tag{1.3}$$

Délka kódového ohraničení udává, jak dlouho se jeden bit podílí na procesu kódování

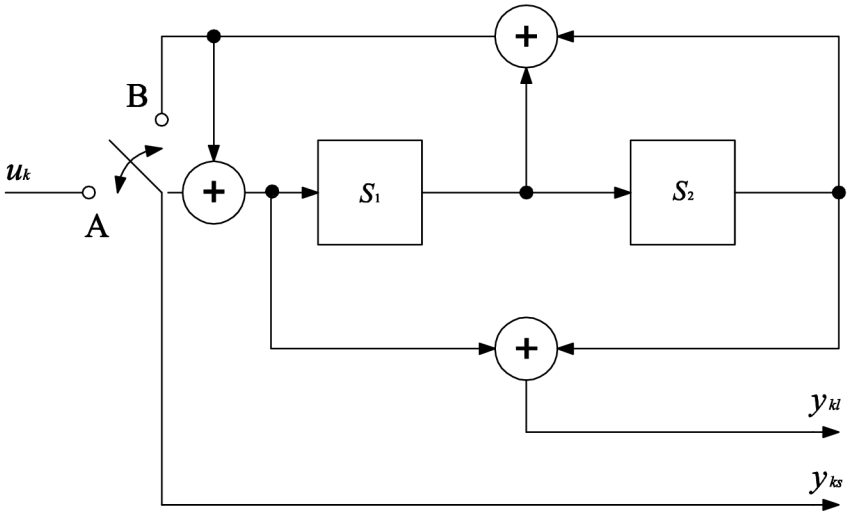
$$K = m + 1, \tag{1.4}$$

kde m je počet paměťových buněk. Všechny stavy, které můžou nastat na paměťových buňkách, jsou zobrazeny na následujícím obr. 1.3. Hodnoty podél přechodů značí výstupní bity kodéru a jsou namapovány do hodnot $+1$ (logická 1) a -1 (logická 0) z důvodu lepší orientace při dekódování.



Obr. 1.3: Mřížový diagram pro RSC (2, 1, 2) kodér.

Po zakódování vstupních bitů je nutné vyprázdnit paměťové buňky. U NRC je řešení jednoduché, a to takové, že se za vstupní posloupnost přidá m nulových bitů.



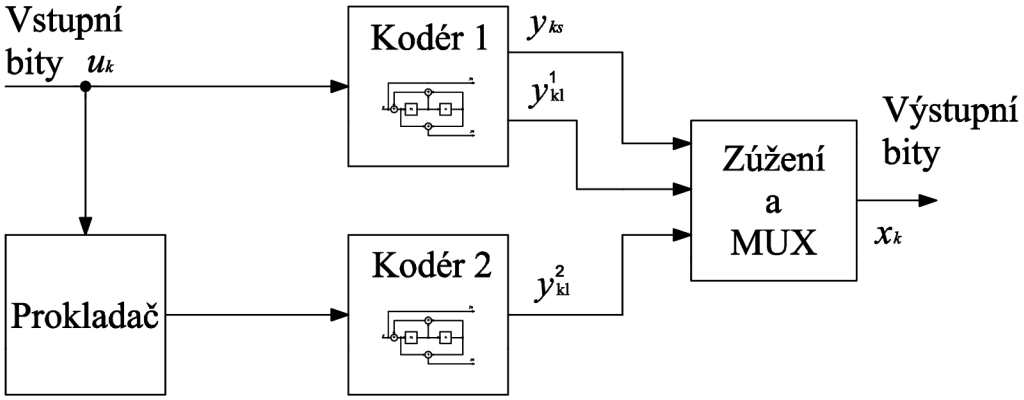
Obr. 1.4: Nulování posuvných registrů RSC (2, 1, 2) kodéru.

Bohužel tento postup není možné aplikovat na RSC kodéry, a to z důvodů jejich zpětné vazby. Vyprazdňování paměťových buněk RSC kodéru je zobrazeno na

obr. 1.4 a je řešeno přidáním přepínače. Pro kódování vstupní posloupnosti je přepínač v poloze A. Pro vyprázdnění paměťových buněk se přesune do polohy B.

1.3 Turbo konvoluční kódování

Blokové schéma turbo kodéru [5] je zobrazeno na obr. 1.5. Jsou zde použity dva totožné kodéry, zpravidla se používají RSC obr. 1.2, mezi kterými je vložen prokladač kap. 1.4. Je možné použít strukturu s více jak dvěma kodéry, ale v této kapitole se budeme zabývat klasickou strukturou o dvou RSC kodérech.



Obr. 1.5: Blokové schéma turbo RSC kodéru.

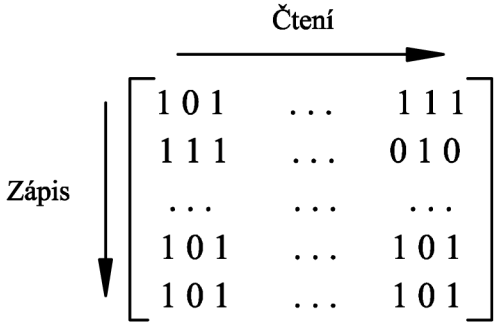
Výstup z obou kodérů je zúžen (děrován) a následně multiplexován (MUX). Obvykle mají oba RSC kodéry informační rychlost 1/2 a dávají jeden systematický a jeden paritní bit na každý vstupní symbol. To znamená, že výstupní posloupnost turbo kodéru x_{kl} bude obsahovat na každý vstupní bit jeden systematický a dva paritní bity tj. y_{1s}, y^1_{1l}, y^2_{1l}, y_{2s}, y^1_{2l}, y^2_{2l}, ..., y_{ks}, y^1_{kl}, y^2_{kl}. Pro tuto výstupní posloupnost má turbo kodér informační rychlost 1/3. Aby byla celková informační rychlost 1/2, musí být část výstupních bitů odstraněna - zúžena (děrována) kap. 1.5. Výstupní posloupnost je zúžena tak, že se ponechají všechny systematické bity a zúženy jsou pouze bity paritní. Po zúžení by výstupní posloupnost turbo kodéru x_{kl} vypadala následovně y_{1s}, y^1_{1l}, y_{2s}, y^2_{2l}, ..., y_{ks}, y^1_{kl}, y_{k+1s}, y^2_{k+1l}. Pokud bychom zúžení použili na bity systematické, došlo by k degradaci výkonu kódu. Z toho důvodu se zúžení (děrování) používá na bity paritní.

1.4 Prokládání

Pro turbokódy se nejčastěji používá blokový nebo pseudonáhodný prokladač [6], který se nachází mezi dvěma kodéry. Slouží k přeskupení vstupních bitů pro druhý kodér, a tím zvyšuje váhu kódového slova.

1.4.1 Blokový prokladač

Blokový prokladač obr. 1.6 je nejpoužívanějším prokladačem v komunikačních systémech. Bity se do něj zapisují od shora dolů a čtení probíhá zprava doleva.



Obr. 1.6: Blokové prokládání.

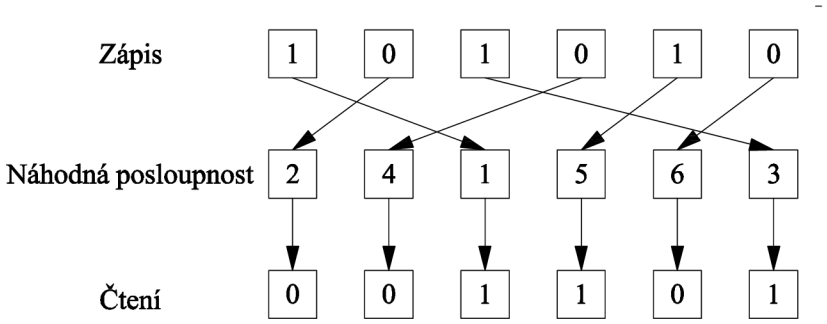
Z obr. 1.6 je zřejmé, že prokladač zapsal do matice [11...11, 01...00] tzn. do sloupců a následně bude bity číst po řádcích [101 ... 111, 111 ... 010].

1.4.2 Pseudonáhodný prokladač

Náhodný prokladač používá náhodnou posloupnost, pomocí které seřadí vstupní bity podle pořadí náhodné posloupnosti. Předpokládejme, že délku vstupní posloupnosti budeme značit F. Na obr. 1.7 je ukázán princip náhodného prokladače pro délku vstupního slova $F = 6$.

Z obr. 1.7 je patrné, že vstupní posloupnost do prokladače byla [101010] a výstupní je [001101]. Matematický zápis prokládání by se dal vyjádřit jako:

$$u'_k = u_k \cdot \Pi, \tag{1.5}$$



Obr. 1.7: Pseudonáhodné prokládání.

kde u'_k je výstupní vektor bitů, Π je matice pseudonáhodného prokladače

$$\Pi = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

a u_k vektor vstupních bitů

$$u_k = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

1.5 Zúžení (děrování)

Výhodou zúžení [8] je snížení počtu bitů v přenášené posloupnosti, a tím i zmenšení přenosové šířky pásma. Nevýhodou je ovšem snížení zabezpečovací schopnosti kódu. Princip spočívá v odstranění bitů z výstupní posloupnosti kodéru podle zúžovací matice **J**. V případě turbo kodérů se zúžují pouze bity paritní y_{kl}, a to z důvodu, že kdybychom zúžovali systematické bity y_{ks}, došlo by k degradaci výkonu kódu.

Princip si ukážeme na následujícím příkladu. Pokud by výstupní posloupnost z kodéru byla

$$\begin{aligned} y_{ks} &= 1010 \\ y_{kl}^1 &= 1111 \\ y_{kl}^2 &= 1111 \end{aligned}$$

a zúžovací matice

$$\mathbf{J} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix},$$

pak v místě, kde se v zúžovací matici nachází 1, bude bit v posloupnosti ponechán, a kde 0, bude bit vymazán. Výstupní posloupnost po zúžení bude

$$\begin{aligned} y_{ks} &= 1\ 0\ 1\ 0 \\ y_{kl}^1 &= 1 - 1 - \\ y_{kl}^2 &= -1 - 1, \end{aligned}$$

kde „−“ značí zúžená místa v posloupnosti. Pro porovnání si ukážeme multiplexovanou posloupnost bez zúžení

$$x_{kl}^1 = 1110111110111$$

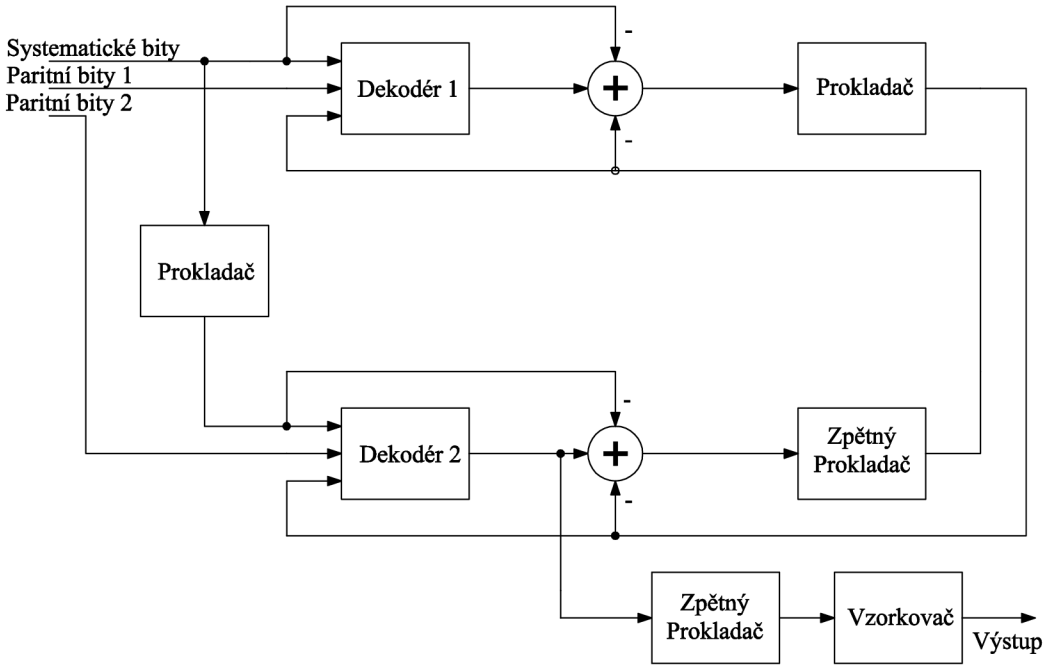
a se zúžením

$$x_{kl}^2 = 11011101.$$

Je tedy zřejmé, že se informační poměr sníží z 1/3 na 1/2. Před dekódováním je nutné provést opačný proces, který na vymazaná místa v posloupnosti vrátí logické 0.

1.6 Turbo konvoluční dekódování

Blokové schéma turbo dekodéru je zobrazeno na obr. 1.8. Propojení dvou dekodérů je přes prokladač podobně jako u kodéru. Jak je zřejmé z obrázku, každý dekodér má tři vstupy. První vstup je pro systematické bity z kodéru, druhý pro paritní bity a třetí pro pravděpodobnostní informace z druhého dekodéru. Tato informace se nazývá jako a-priori. Oba dekodéry mají tzv. měkké výstupy pro dekódované bity. To znamená, že dekódovaným bitům poskytují také pravděpodobnost správné hodnoty výstupního bitu. Měkký výstup je reprezentován takzvaným LLR (logaritmicko pravděpodobnostní poměr – Log Likelihood Ratio). Polarita LLR rozhoduje o dekódovaném bitu a hodnota udává pravděpodobnost správnosti rozhodnutí.



Obr. 1.8: Blokové schéma turbo dekodéru.

1.6.1 Log Likelihood Ratios

LLR slouží ke zjednodušení procházení zprávy v iterativním dekódování z jednoho dekodéru k druhému, a proto je nezbytnou součástí turbo kódu. LLR datového

bitu se značí $L(u_k)$ a je definovaný jako logaritmus podílu pravděpodobností dvou možných hodnot bitů.

$$L(u_k) = \ln\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right). \tag{1.6}$$

Možné hodnoty bitů u_k jsou +1 a –1 místo klasických 1 a 0. Tato změna je z důvodu jednoduššího matematického vyjádření. Když se hodnota LLR $L(u_k) \approx 0$, dostaneme $P(u_k = +1) \approx P(u_k = -1) \approx 0,5$ ($\approx$ zn. aproximace) a z toho plyne, že si nemůžeme být jistí hodnotou u_k. Naopak, když je $L(u_k) \gg 0$, dostaneme $P(u_k = +1) \gg P(u_k = -1)$, a tedy jistotu, že hodnota $u_k = +1$. Pro LLR $L(u_k)$ je možné vypočítat pravděpodobnost $u_k = +1$ a $u_k = -1$ vzorce odvozeného z [5]:

$$P(u_k = \pm 1) = \left(\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}}\right) \cdot e^{\pm L(u_k)/2}. \tag{1.7}$$

Stejně tak, jak je důležitý LLR $L(u_k)$ založený na nepodmíněných pravděpodobnostech, tak je také důležitý LLR založený na podmíněných pravděpodobnostech. V teorii kanálového kódování se zajímáme o základní nebo podmíněnou pravděpodobnost pro $u_k = \pm 1$ na přijaté posloupnosti $\underline{y}$, a proto můžeme použít podmíněný LLR $L(u_k \mid \underline{y})$ definovaný jako [5]:

$$L(u_k \mid \underline{y}) = \ln\left(\frac{P(u_k = +1 \mid \underline{y})}{P(u_k = -1 \mid \underline{y})}\right). \tag{1.8}$$

Podmíněná pravděpodobnost $P(u_k = +1 \mid \underline{y})$ je známá jako a-posteriori pravděpodobnost dekódovaného bitu u_k.

Jestliže budeme předpokládat, že přenášený bit $x_k = \pm 1$ je poslán přes Gaussův nebo kolísající (slábnoucí) kanál používající BPSK modulaci, pak můžeme zapsat [5]:

$$\begin{aligned} L(y_k \mid x_k) &= \ln\left(\frac{P(y_k \mid x_k = +1)}{P(y_k \mid x_k = -1)}\right) \\ &= L_c y_k, \end{aligned} \tag{1.9}$$

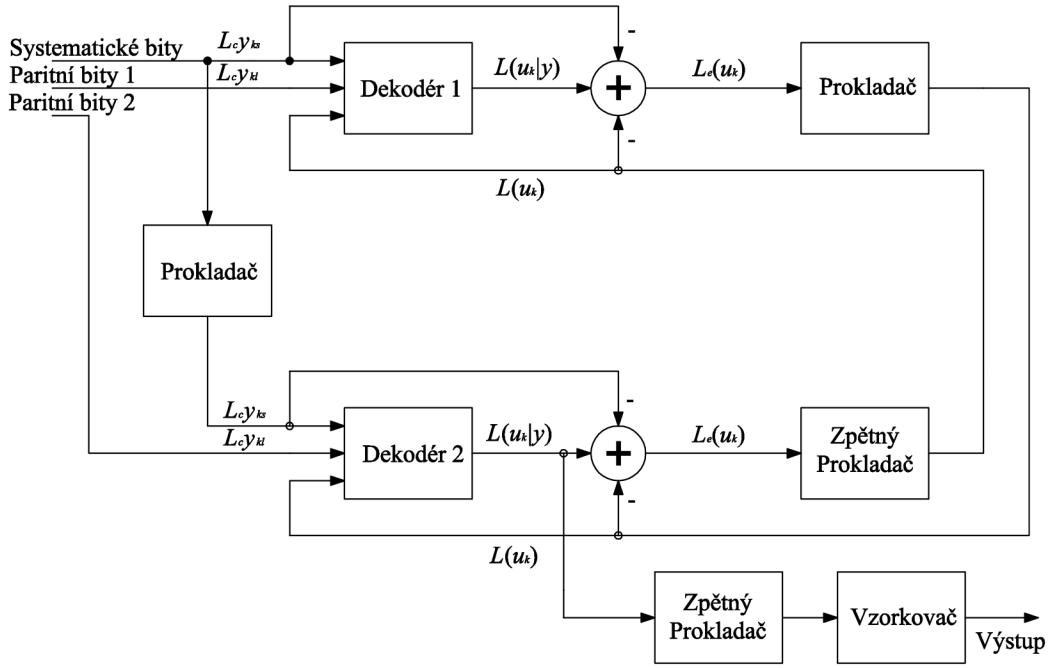
kde

$$L_c = 4a\frac{E_b}{2\sigma^2}, \tag{1.10}$$

který nazýváme jako spolehlivost přenosového kanálu, kde E_b je střední hodnota energie na jeden bit, a je amplituda a σ rozptyl šumu.

1.6.2 Iterativní turbo dekódování

Nyní si popíšeme, jakým způsobem funguje iterativní dekódování. Na obr. 1.9 je znázorněno schéma dekodéru a v něm popsány vstupy a výstupy jednotlivých bloků.



Obr. 1.9: Schéma turbo RSC dekodéru.

První dekodér v první iteraci příjme posloupnost $L_c\underline{y}^{(1)}$ z přenosového kanálu, která obsahuje systematické bity $L_c y_{ks}$ a paritní bity $L_c y_{kl}$ z prvního kodéru. Obvykle je přijata pouze polovina paritních bitů, protože tyto bity jsou zúženy ve vysílači. Proto musí dekodér do měkkého výstupu kanálu $L_c y_{kl}$ vložit nuly na zúžená místa. Nyní může začít dekodér zpracovávat měkké vstupy z kanálu a produkovat odhad $L_{11}(u_k \mid \underline{y})$ LLR datových bitů $u_k, k = 1, 2 \ldots N$. Dolní index symbolu $L_{11}(u_k \mid \underline{y})$ značí a-posteriorní informaci LLR v první iteraci prvního dekodéru. V této první

iteraci nemá dekodér a-priori informaci o bitech, a proto bude hodnota $L(u_k) = 0$, která odpovídá a-priori pravděpodobnosti $0,5$. Nyní začne pracovat druhý dekodér. Ten příjme posloupnost $L_c\underline{y}^{(2)}$, která obsahuje systematické bity pro první dekodér, které prochází přes prokladač a paritní bity z druhého kodéru. Pokud byly tyto paritní bity před vstupem do přenosového kanálu zúženy, musí druhý dekodér do této posloupnosti vložit nuly. Kromě vstupních hodnot $L_c\underline{y}^{(2)}$ příjme druhý dekodér a-priori LLR $L(u_k)$, který mu poskytuje první dekodér z LLR $L_{11}(u_k \mid \underline{y})$. Jak je zřejmé z obr. 1.9, vnější informace $L_e(u_k)$ z prvního dekodéru je upravena prokladačem tak, aby odpovídala pořadí bitů vstupujících do druhého dekodéru, který použije tuto proloženou informaci a přijatou posloupnost $L_c\underline{y}^{(2)}$ k výpočtu a-posteriori LLR $L_{12}(u_k \mid \underline{y})$. Nyní podle vzorce [5]:

$$L_e(u_k) = L(u_k \mid \underline{y}) - L(u_k) - L_c y_{ks}, \tag{1.11}$$

se od výstupu dekodéru $L(u_k \mid \underline{y})$ odečte systematický měkký vstup $L_c y_{ks}$ a a-priori informace $L_{(uk)}$ z předchozího dekodéru. Vypočítaná hodnota bude vnější informace $L_e(u_k)$, použije se jako a-priori informace pro první (následující) dekodér v druhé iteraci. Tímto končí první iterace pro oba dekodéry.

V druhé iteraci opět první dekodér zpracovává přijatou posloupnost $L_c\underline{y}^{(1)}$, ale tentokrát má k dispozici a-priori informaci, která je zpětně proložená vnější informace $L_e(u_k)$ vypočítaná druhým dekodérem z a-posteriori LLR $L_{12}(u_k \mid \underline{y})$, a tedy první dekodér může vypočítat přesnější a-posteriori LLR $L_{21}(u_k \mid \underline{y})$. Druhá iterace pak pokračuje ve druhém dekodéru, který použije přesnější a-posteriori LLR $L_{21}(u_k \mid \underline{y})$ z prvního dekodéru, který vypočítal pomocí vzorce 1.11 přesnější a-priori informaci $L(u_k)$. Tato informace se použije společně s přijatou posloupností $L_c\underline{y}^{(2)}$ k výpočtu $L_{22}(u_k \mid \underline{y})$, ze kterého se následně počítá $L_e(u_k)$ pro následující (první) dekodér.

Když je série iterací dokončena, výstup turbo dekodéru je dán zpětným proložením a-posteriori LLR druhého dekodéru $L_{i2}(u_k \mid \underline{y})$, kde i je počet použitých iterací. Znaménka a-posteriori posloupnosti dávají výstup tvrdého rozhodnutí t.j. $+1$ nebo -1.

1.6.3 Matematický popis SOVA (RSC)

Tato kapitola popisuje Viterbiho algoritmus, který je uváděný jako SOVA (měkký výstup Viterbiho algoritmu – Soft-Output Viterbi Algorithm). Pro dekódování turbo kódů má tento algoritmus dvě modifikace. První modifikace je upravení metriky cesty tak, aby zohlednila a-priori informaci při výběru cesty s maximální pravděpodobností v mřížovém diagramu. Druhá modifikace algoritmu spočívá v měkkém výstupu v podobě a-posteriori LLR $L(u_k \mid \underline{y})$ pro každý dekódovaný bit.

První modifikace posuzuje posloupnost stavů $\underline{s}_k^s$, která dává stavy podél přežívající cesty ve stavu $S_k = s$ v k-tém kroku v mřížovém diagramu. Pravděpodobnost, že vybudovaná cesta procházející mřížovým diagramem je správná, je dána [5]:

$$P(\underline{s}_k^s \mid \underline{y}_{j \leq k}) = \frac{P(\underline{s}_k^s \wedge \underline{y}_{j \leq k})}{P(\underline{y}_{j \leq k})}. \tag{1.12}$$

Pravděpodobnost přijaté posloupnosti $\underline{y}_{j \leq k}$ až do k-tého přechodu je konstantní pro všechny cesty $\underline{s}_k$ v mřížovém diagramu až do k-té pozice. Pravděpodobnost, že cesta $\underline{s}_k^s$ je správná, je úměrná $P(\underline{s}_k^s \wedge \underline{y}_{j \leq k})$. Proto by metrika měla být definována tak, že největší metrika bude mít největší $P(\underline{s}_k^s \wedge \underline{y}_{j \leq k})$. Metrika by měla být snadno vypočitatelná rekursivním způsobem, kde jdeme v mřížovém diagramu z kroku $(k-1)$ do k-tého kroku. Vhodná metrika pro cestu $\underline{s}_k^s$ je tedy odvozená z [5]:

$$M(\underline{s}_k^s) = M(\underline{\hat{s}}_{k-1}^{\hat{s}}) + \frac{1}{2} u_k L(u_k) + \frac{L_c}{2} \sum_{t=1}^{n} y_{kl} x_{kl}. \tag{1.13}$$

Nyní zmíníme druhou modifikaci algoritmu a to měkký výstup. V binárním mřížovém diagramu jsou dvě cesty jdoucí do stavu $S_k = s$ v k-tém kroku. Modifikace Viterbiho algoritmu vezme a-priori informaci, vypočítá metriku těchto dvou cest podle vzorce 1.13 a vyřadí cestu s menší metrikou. Když obě cesty $\underline{s}_k^s$ a $\underline{\hat{s}}_k^s$ jdoucí do stavu $S_k = s$ mají metriku $M(\underline{s}_k^s)$ a $M(\underline{\hat{s}}_k^s)$ a je zvolena cesta s vyšší metrikou $\underline{s}_k^s$ za přežívající, můžeme počítat rozdíl metrik Δ_k^s těchto cest jako [5]:

$$\Delta_k^s = M(\underline{s}_k^s) - M(\underline{\hat{s}}_k^s) \geq 0. \tag{1.14}$$

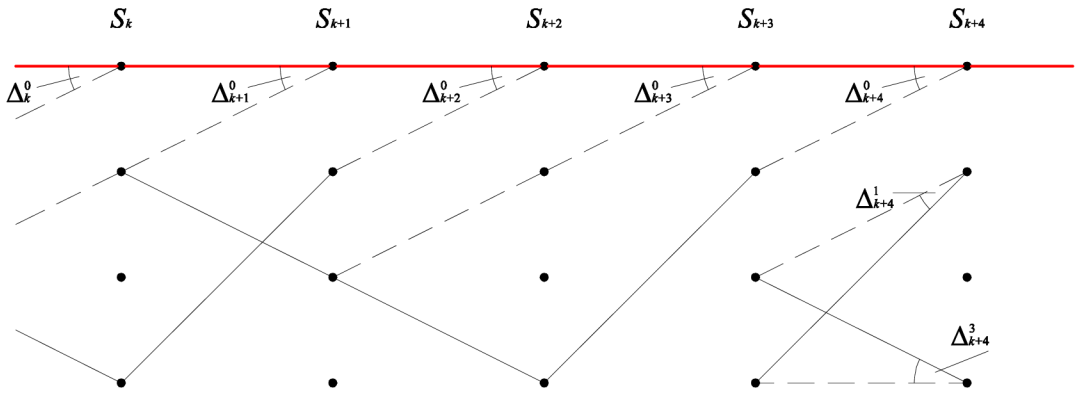
Pravděpodobnost, že se jedná o správné rozhodnutí mezi zvolenou přežívající cestou $\underline{s}_k^s$ a vyřazenou cestou $\underline{\hat{s}}_k^s$ je dána vzorcem odvozeným z literatury [5]:

$$P(\text{správné rozhodnutí v } S_k = s) = \frac{P(\underline{s}_k^s)}{P(\underline{s}_k^s) + P(\underline{\hat{s}}_k^s)} \tag{1.15}$$

a LLR správného rozhodnutí je dáno jako:

$$\begin{aligned} L(\text{správné rozhodnutí v } S_k = s) &= \ln\left(\frac{P(\text{správné rozhodnutí v } S_k = s)}{1 - P(\text{správné rozhodnutí v } S_k = s)}\right) \\ &= \Delta_k^s. \end{aligned} \tag{1.16}$$

Na obr. 1.10 je znázorněn zjednodušený mřížový diagram pro RSC kód, kde $K = 3$, ve kterém jsou vyznačeny rozdíly v metrikách jednotlivých cest.



Obr. 1.10: Zjednodušený mřížový diagram pro RSC dekódování.

Když dojdeme na konec mřížového diagramu a zjistíme ML (maximálně pravděpodobná – Maximum Likelihood) cestu, je nutné najít LLR. Ten určuje spolehlivost o rozhodování bitů okolo ML cesty. Viterbiho algoritmus ukazuje, že všechny přežívající cesty v pozici l vyšly v předchozích pozicích z této cesty a v pozici l se do ní vracejí. Tato předchozí pozice může nabýt δ přechodů před pozicí, kde δ je zpravidla nastavena na pětinásobek omezení délky konvolučního kódu. Proto hodnota bitu u_k spojeného s přechodem ze stavu $S_{k-1} = s'$ do stavu $S_k = s$ na ML cestě může být rozdílná, když místo ML cesty Viterbiho algoritmus vybere některou cestu

sloučenou s ML cestou o δ přechodů později tj. o $k+\delta$ stavů v mřížovém diagramu. V případě, že algoritmus vybere některou z cest sloučenou s ML cestou, hodnotu u_k by to nemělo ovlivnit, protože tato cesta se bude lišit od ML cesty od přechodu z $S_{k-1} = s'$ do $S_k = s$. Když vypočítáme LLR pro bit u_k, SOVA musí vzít v úvahu pravděpodobnost cesty slučující se s ML cestou v kroku k do kroku $k+\delta$. Porovnáním rozdílů v metrikách $\Delta_i^{s_i}$ pro všechny stavy s_i podél ML cesty ze stavu $i = k$ do $i = k+\delta$ [5]:

$$L(u_k \mid \underline{y}) \approx u_k \min_{i=k\cdots k+\delta \;\; u_k \neq u_k^i} \Delta_i^{s_i}, \tag{1.17}$$

kde u_k je hodnota bitu daného ML cestou a u_k^i je hodnota bitu daného cestou, která se spojila s cestou ML a byla zrušena ve stavu i. Minimalizace ve vzorci 1.17 se použije jen pro cesty slučující se s ML cestou, které dávají rozdílnou hodnotu pro bit u_k, když ji zvolí jako přežívající. Cesty, které daly stejnou hodnotu u_k jako ML cesta, nemají vliv na rozhodnutí.

Pro objasnění těchto operací se znovu podíváme na obr. 1.10, který ukazuje zjednodušený mřížový diagram pro RSC kód, kde $K = 3$. Na tomto obrázku plné čáry vyznačují vstupní bit $+1$ a čerchované vstupní bit -1. Předpokládejme, že cesta pro samé nuly nám neprezentuje ML cestu, která je zobrazena tučnou červenou čarou. Jsou zde ukázány cesty, které se slučují s ML cestou. Z obrázku je zřejmé, že ML cesta dává hodnoty $u_k - 1$, zatímco cesty slučující se s ML cestou v pozici $S_k,\ S_{k+1},\ S_{k+3}$ a S_{k+4} dávají hodnotu bitu u_k $+1$. Jestliže pro jednoduchost budeme předpokládat, že hodnota $\delta = 4$, pak ze vzorce 1.17 pro LLR $L(u_k \mid \underline{y})$ dostaneme –1 krát minimální rozdíl metriky $\Delta_k^0,\ \Delta_{k+1}^0,\ \Delta_{k+3}^0$ a Δ_{k+4}^0.

SOVA je implementován následujícím způsobem. V každém stavu v každé pozici v mřížovém diagramu je metrika $M(\underline{s}_k^s)$ počítána pro obě cesty jdoucí do stavu použitím vzorce 1.13. Cesta s větší metrikou je pro tento stav zvolena jako přežívající a ukazatel metriky uložen tak, jak to dělá Viterbiho algoritmus. Nicméně, aby poskytoval spolehlivě dekódované bity, ukládá si také hodnotu $L(u_k|\underline{y})$, kterou vypočítá podle vzorce 1.17. Takto je uložen rozdíl metrik mezi přežívající a vyřazenou cestou společně s binárním vektorem skládající se z $\delta + 1$ bitů, který indikuje

jestli by vyřazená cesta mohla dát stejnou posloupnost bitů u_l pro $l = k$, zpět k $l = k - \delta$, jako cesta přežívající. Tato série bitů se nazývá update sekvence a je dána výstupem modulo 2 mezi předchozím $\delta + 1$ dekódovaným bitem podél přežívající a zrušené cesty. Když SOVA identifikuje ML cestu, jsou uložené update sekvence a metrické rozdíly podél cesty použity k výpočtu hodnoty $L(u_k|\underline{y})$.

1.7 Příklad turbo konvolučního dekódování pomocí SOVA

V této kapitole budeme rozebírat příklad dekódování turbo kódů používajících SOVA [4][5][6][7]. Použijeme jednoduchý turbo kód s informačním poměrem 1/2, který používá $K = 3$ RSC kód s vytvářecím mnohočlenem 7 a 5 (osmičková soustava). Kombinací dvou RSC kodérů a náhodného prokladače obr. 1.5 dostaneme jednoduchý turbo kodér. Paritní bity z těchto kodérů jsou zúženy (děrovány) tak, že z prvního kodéru jsou přeneseny první, třetí, pátý, sedmý, devátý paritní bit a z druhého kodéru druhý, čtvrtý, šestý a osmý paritní bit. První kodér použije k ukončení dva bity k vynulování paměťových buněk kodéru. Proto bude přenesená posloupnost obsahovat 9 systematických a 9 paritních bitů. Systematické bity tedy obsahují 7 vstupních a 2 ukončovací bity. Z paritních bitů je přenášeno 5 z prvního kodéru a 4 z druhého kodéru.

Stavový diagram pro RSC kód je znázorněn na obr. 1.3, spojitá čára značí vstupní bit -1 (0) a čerchovaná značí vstupní bit +1 (1). Čísla podél čar znázorňují výstupní bity kodéru, pro daný vstupní bit (přechod), kde první bit je systematický a druhý bit je paritní.

Pro jednoduchost budeme předpokládat, že všechny vstupní bity budou –1. Pro přenos použijeme BPSK modulaci, která užívá k přenosu symboly –1 a +1. Detailní přehled vstupních, výstupních a přenesených bitů je znázorněn v tab. 1.1. Všimněte si, že paritní bity z každého kodéru jsou zúženy na polovinu.

Nyní si ukážeme rozdíl mezi iterativním dekódováním a dekódováním konvolučních kódů. Nejprve budeme přijatou posloupnost v tab. 1.1 dekódovat konvolučním dekodérem, který používá Viterbiho algoritmus. Předpokládejme poloviční poměr

Vstupní bit	Systematický bit	Paritní bity kodér 1	kodér 2	Přenášená posloupnost	Přijatá posloupnost
–1	–1	–1	–	–1, –1	–1,9, –0,6
–1	–1	–	–1	–1, –1	–0,9, –1,8
–1	–1	–1	–	–1, –1	–2,1, –0,7
–1	–1	–	–1	–1, –1	+1,4, +0,2
–1	–1	–1	–	–1, –1	+1,6, –0,8
–1	–1	–	–1	–1, –1	–1,3, –1,1
–1	–1	–1	–	–1, –1	–0,5, –1,3
–	–1	–	–1	–1, –1	–2,7, –2,1
–	–1	–1	–	–1, –1	–1,2, –1,2

Tab. 1.1: Vstupní a přenášené bity pro příklad turbo dekódování.

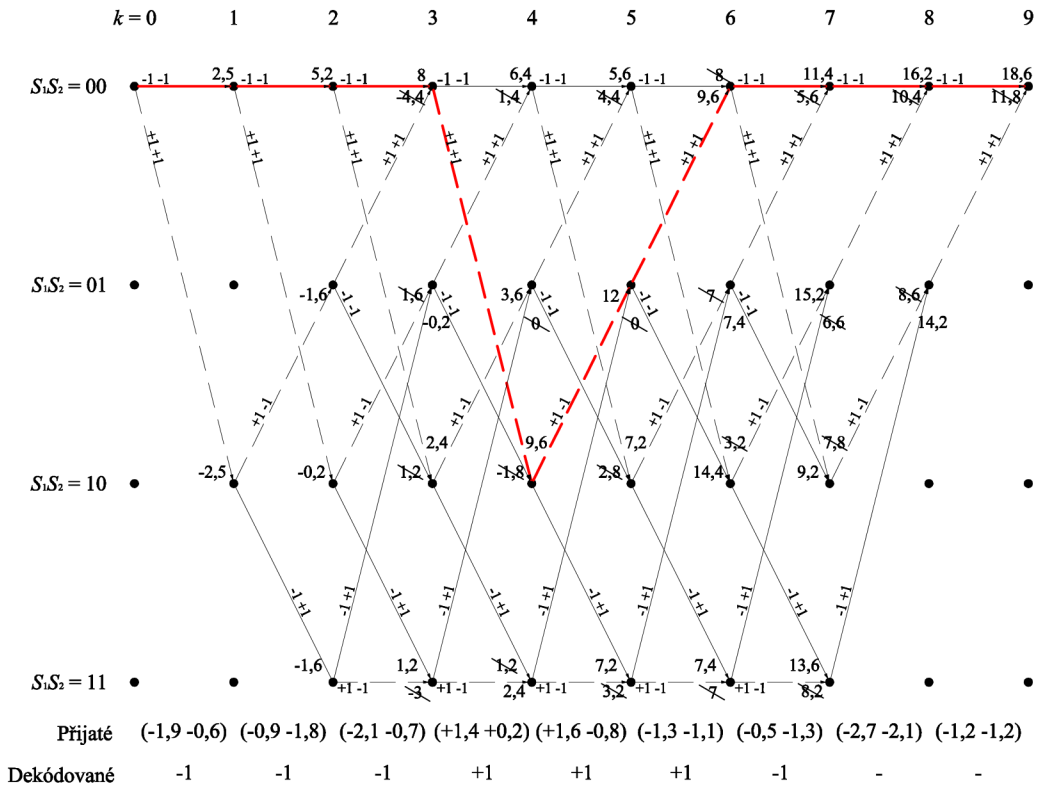
$K = 3$ RSC kódů užívaný jako běžný konvoluční kód, kterým zakódujeme vstupní posloupnost sedmi –1. Jestliže by jsme použili k vynulování paměti kodéru dvě –1, pak by se naše přenášená posloupnost skládala z osmnácti –1. Přijatou posloupnost bude Viterbiho algoritmus dekódovat pomocí mřížového diagramu znázorněného na obr. 1.11. Metrika znázorněná v tomto obrázku je dána vzájemnou korelací (cross corelation) přijatých a očekávaných kanálových posloupností pro danou cestu a Viterbiho algoritmus maximalizuje tyto metriky k nalezení ML cesty, která je znázorněna tučnou čarou. Všimněte si, že v každém stavu v mřížovém diagramu, kde se slučují dvě cesty je cesta s menší metrikou vyřazena a její hodnota je přeškrtnuta. Z obrázku je zřejmé, že Viterbiho algoritmus udělal nesprávné rozhodnutí v pozici $k = 6$ a zvolí jinou cestu než nenulovou jako přežívající. To má za následek, že tři ze sedmi bitů budou mít nesprávnou hodnotu +1.

Zde je tedy vidět, že Viterbiho dekódování pro RSC kód by udělalo tři chyby pro přijatou zprávu. Nyní si podrobně ukážeme, jakým způsobem pracuje iterativní turbo dekodér pro stejnou kanálovou posloupnost. Uvažujme nejdříve první dekodér v první iteraci. Ten nepoužívá SOVA pouze k určení nejpravděpodobnějšího vstupního bitu, ale také k určení LLR tohoto bitu. Popíšeme si, jak SOVA počítá tyto LLR pro kanálové hodnoty dané v tab. 1.1.

Metriky pro SOVA jsou dány vzorcem 1.13, který je pro zjednodušení opakován [5]:

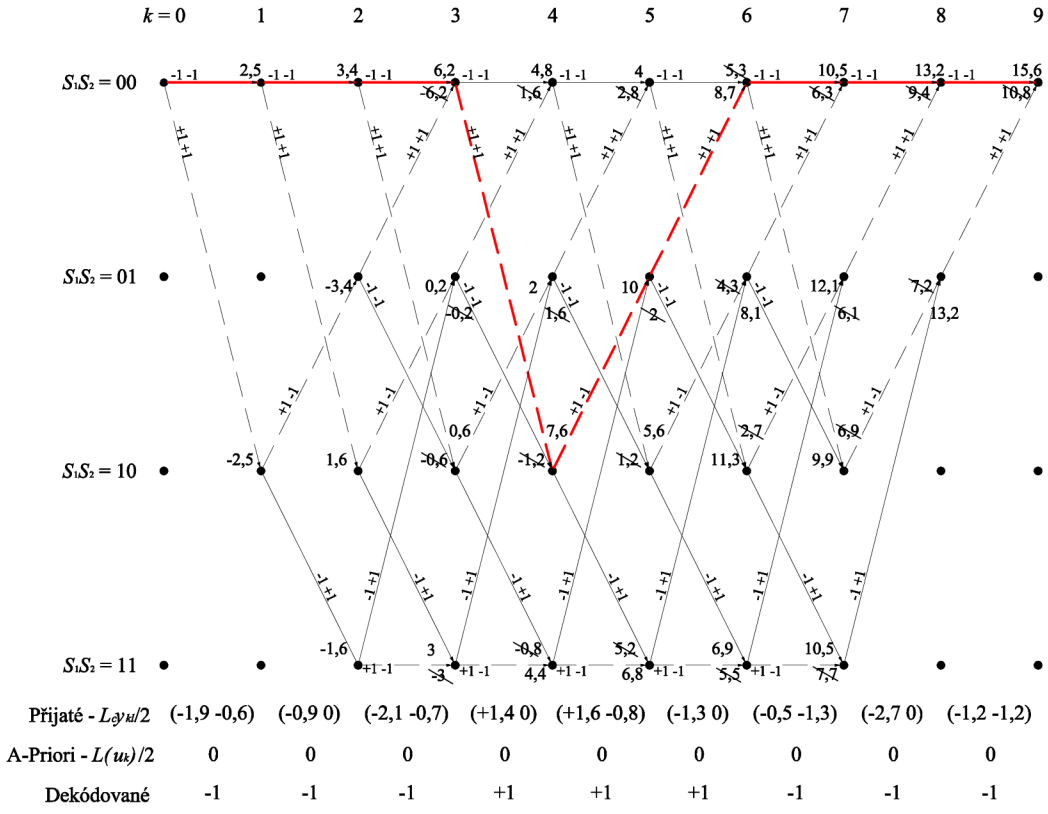
$$M(\underline{s}_k^s) = M(\underline{s}_{k-1}^{\grave{s}}) + \frac{1}{2}u_k L(u_k) + \frac{L_c}{2}\sum_{t=1}^{n} y_{kl}x_{kl}. \tag{1.18}$$

Kde $M(\underline{s}_{k-1}^{\grave{s}})$ je metrika přežívající cesty skrz stav $S_{k-1} = \grave{s}$ v pozici $k-1$ v mřížovém diagramu, u_k je vstupní bit kodéru, x_{kl} přenášená posloupnost (výstup z kodéru) pro daný přechod a y_{kl} je přijatá posloupnost z přenosového kanálu spojená s tímto přechodem. L_c je spolehlivost přenosového kanálu definovaná ve vzorci 1.10. V první iteraci nemáme pro první dekodér a-priori informaci, a proto je hodnota $L(u_k) = 0$, pro všechny pozice k, která odpovídá a-priori pravděpodobnosti $0,5$. Přijatá posloupnost daná v tab. 1.1 je odvozená z posloupnosti přenosového kanálu (kde $E_b = 1$) přidáním AWGN (aditivní bílý Gaussův šum – Aditive White Gaussian Noise) s rozptylem $\sigma = 1$. Pokud bude kolísání amplitudy $a = 1$, pak podle vzorce 1.10 bude hodnota spolehlivosti přenosového kanálu $L_c = 2$.



Obr. 1.11: Mřížový diagram Viterbiho dekódování přijaté posloupnosti znázorněné v tab. 1.1 .

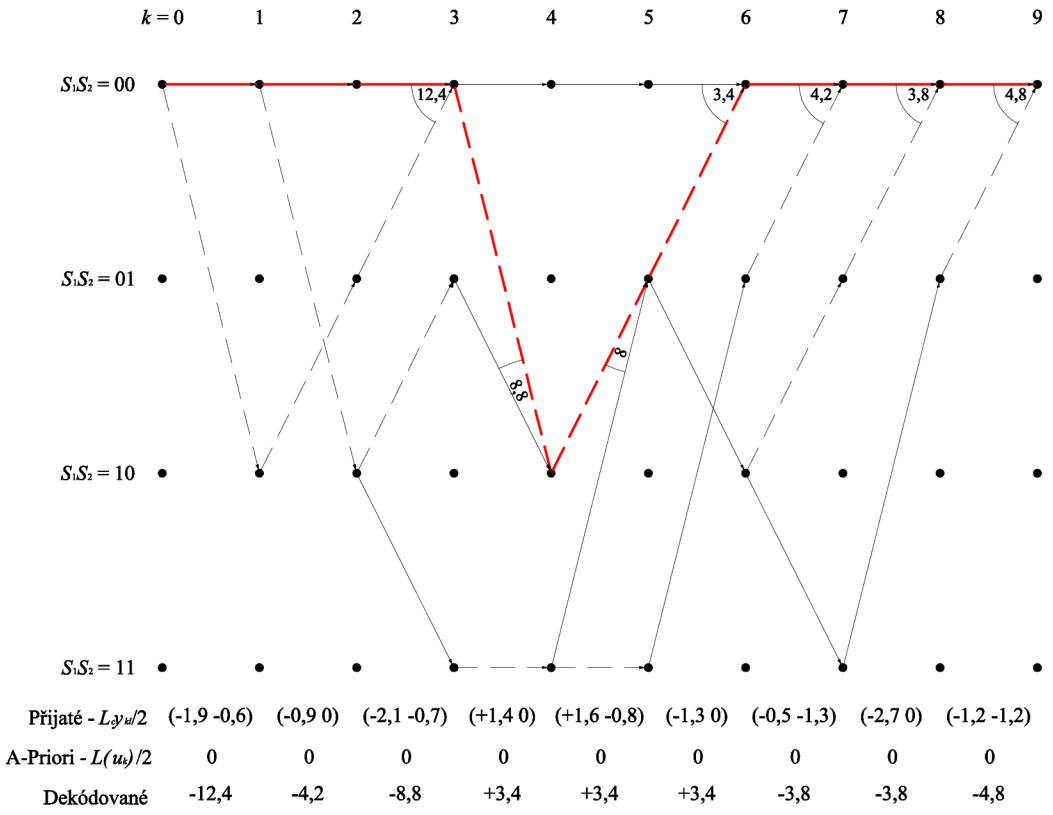
Obr. 1.12 znázorňuje mřížový diagram pro první dekodér v první iteraci. Vzhledem k tomu, že paritní bity z prvního kodéru byly zúženy, tak přijatá hodnota pro druhý, čtvrtý, šestý a osmý bit bude 0. A-priori a hodnoty z přenosového kanálu jsou znázorněny v obr. 1.12 jako $L(u_k)/2$ a $L_c y_{kl}/2$ z důvodu, aby bylo možné hodnoty metrik dané vzorcem 1.18 vypočítat jednoduchým sčítáním a odečítáním zobrazených hodnot. Dále máme hodnoty $L(u_k) = 0$ a $L_c = 2$. Metriky jsou, jako v případe Viterbiho algoritmu, znovu dány vzájemnou korelací očekávané a přijaté posloupnosti. Protože bylo použito zúžení, metrické hodnoty v obr. 1.12 se neshodují s metrikami v obr. 1.11.



Obr. 1.12: Mřížový diagram SOVA dekódování pro první dekodér v první iteraci.

Výpočty pro SOVA jsou odlišné od výpočtů Viterbiho algoritmu, protože v případe SOVA můžeme mít více než jednu vznikající větev v daném stavu v mřížovém diagramu. Jako příklad si uvedeme obr. 1.12, ve stavu $S_1S_2 = 01$ a pozici $k = 5$, kde obě cesty vycházející z tohoto stavu přežijí, a to z důvodu, že tyto cesty ve-

dou k dalším stavům v následující pozici a tam jsou opět zvoleny jako přežívající. Konkrétněji, cesta pro vstupní bit 0 (spojitá čára) vede do stavu $S_1S_2 = 10$ a má celkovou metriku cesty $11,3$. Tato metrika se porovná s metrikou jiné cesty $(2,7)$ vedoucí ze stavu $S_1S_2 = 00$ do stavu $S_1S_2 = 10$. Jako „vítězná“ je zvolena cesta s vyšší metrikou. Podobně je tomu u cesty vedoucí ze stavu $S_1S_2 = 01$ v kroku $k = 5$ do stavu $S_1S_2 = 00$ v kroku $k = 6$, která je určena pro vstupní bit $+1$ (čerchovaná čára) a její celková metrika je $8,7$. Tato cesta je následně porovnána s hodnotou jiné cesty $(5,3)$, která přichází ze stavu $S_1S_2 = 00$, a je následně zvolena jako „vítězná“. I přes tuto výhodu, že je SOVA schopen mít více než jednu vznikající větev v daném uzlu, je zvolená ML cesta (tlustá červená čára), znázorněná v obr. 1.12, stejná jako ML cesta zvolená Viterbiho algoritmem v obr. 1.11. V obou případech došlo ke třem chybám v dekódování.



Obr. 1.13: Zjednodušený mřížový diagram SOVA dekódování pro první dekodér v první iteraci.

Nyní se budeme věnovat nalezení LLR pro dekódované bity konkrétní ML cesty. Obr. 1.13 je zjednodušená verze mřížového diagramu z obr. 1.12, který znázorňuje pouze ML cestu a cesty, které s touto cestou byly sloučené a vyřazené. Také ukazuje rozdíl v metrikách, označované jako Δ_k^s, mezi ML cestou a zrušenými cestami. Tyto rozdíly v metrikách, společně s update sekvencí signalizující, které bity přežívající a zrušené cesty uložené SOVA pro každý uzel v každé pozici v mřížovém diagramu, mohou dát jiné hodnoty dekódovaných bitů. Když je určena ML cesta, algoritmus použije tyto uložené hodnoty podél ML cesty k nalezení LLR pro každý dekódovaný bit. V tab. 1.2 jsou znázorněny uložené hodnoty pro mřížový diagram na obr. 1.12 a obr. 1.13. Výpočet dekódovaných LLR zobrazených v tab. 1.2 je popsán v následujícím odstavci.

Pozice v MD k	Dekódovaný bit u_k	Rozdíl metrik Δ_k^s	Update sekvence	Výstup dekodéru LLR $L(u_k\|\underline{y})$
1	–1	–	–	–12,4
2	–1	–	–	–4,2
3	–1	12,4	111	–8,8
4	+1	8,8	1110	+3,4
5	+1	8	10010	+3,4
6	+1	3,4	111000	+3,4
7	–1	4,2	1100010	–3,8
8	–1	3,8	11100000	–3,8
9	–1	4,8	100100000	–4,8

Tab. 1.2: Výstup SOVA pro první dekodér v první iteraci.

Všimněte si, že pro pozice $k = 1$ a $k = 2$ v mřížovém diagramu nejsou v tab. 1.2 uvedeny rozdíly metrik a update sekvence, protože jak je zřejmé z obr. 1.12 a obr. 1.13, nejsou v těchto pozicích s ML cestou sloučeny žádné cesty. Hodnota 1 pro update sekvenci značí, že ML a zrušená slučující se cesta dávají rozdílné hodnoty pro dekódovaný bit a naopak. V případě, že budeme mřížový diagram procházet zleva, narazíme na nejbližší pozici k, kde se bude slučovat ML cesta s cestou vyřazenou. Dekódovaný bit u_k náležící tomuto přechodu, kde se tyto cesty slučují, nazýváme MSB (nejvíce významný bit – Most Significant Bit). Druhý nejvýznamnější bit bude u_{k-1} atd., dokud nenarazíme na bit u_1, který nazýváme jako LSB (nejméně významný bit – Least Significant Bit). Náš RSC kód má vždy dvě cesty slučující se

v pozici k a tyto dvě cesty mají vždy rozdílnou hodnotu dekódovaného bitu u_k. Z tohoto důvodu bude vždy hodnota update sekvence pro MSB 1. V našem příkladu mají update sekvence různé délky, protože kódovaná posloupnost je velmi krátká. V praxi bývá tato délka $\delta + 1$ dlouhá, kde δ bývá nastavena na pětinásobek délky kódového ohraničení konvolučního kódu (v našem případě by byla 15).

Nyní si vysvětlíme, jak SOVA využívá uložené update sekvence a rozdíl v metrikách podél ML cesty k vypočítání LLR pro dekódované bity. Vzorec 1.17, který je pro jednodušší orientaci zopakován v 1.19 říká, že dekódovaná a-posteriori LLR $L(u_k|\underline{y})$ pro bit u_k je dána minimálním rozdílem metrik slučujících se cest podél ML cesty [5]

$$L(u_k \mid \underline{y}) \approx u_k \min_{i=k\cdots k+\delta \;\; u_k \neq u_k^i} \Delta_i^{s_i}, \tag{1.19}$$

Toto minimum je pouze převzaté z metrických rozdílu pro pozici $i = k, k+1, \ldots, k+\delta$, kde hodnota u_k^i, bitu u_k, daná slučující se cestou s ML cestou v pozici i, je rozdílná od hodnoty bitu dané ML cestou. Uložené update sekvence v pozici k podél ML cesty se značí jako $\underline{e}_k$. Pro druhý MSB se bude update sekvence značit jako $\underline{e}_{k+1}$ atd. až po $(\delta+1)$-tý bit $\underline{e}_{k+\delta}$. Jak už bylo řečeno, v tomto příkladu update sekvence nedosahují těchto délek, ale princip je stále stejný. Pro ukázku vezmeme bit u_4 a určíme dekódované LLR $L(u_4|\underline{y})$. Algoritmus zkoumá MSB v update sekvenci $\underline{e}_4$ ve čtvrtém řádku tab. 1.2, druhý MSB v $\underline{e}_5$ (řádek 5) atd. až k šestému MSB v $\underline{e}_9$ (řádek 9). Zde zjistíme, že slučované cesty v pozicích $k = 4$ a 6 v mřížovém diagramu dávají rozdílné hodnoty od ML cesty pro bit u_4. Hodnotu LLR ($L(u_4|\underline{y})$) pro tento bit se tedy vypočítá podle vzorce 1.19 jako hodnota daná ML cestou (+1) násobená minimálním rozdílem metrik uložených v pozicích 4 a 6 ($8, 8$ a $3, 4$). Hodnota LLR bude tedy $L(u_4|\underline{y}) = +3, 4$. Pro následující bit se LLR vypočítá následovně. Zvolíme první MSB v $\underline{e}_5$ (řádek 5), druhý MSB v $\underline{e}_6$ (řádek 6) atd. až pátý MSB v $\underline{e}_9$ (řádek 9). Tyto MSB znázorňují rozdílné hodnoty pro tento bit dané zrušenou cestou v pozici $k = 5$ a 6. Protože metrický rozdíl ve stavu $k = 6$ $(3, 4)$ je menší než metrický rozdíl ve stavu $k = 5$ (8), bude $L(u_5|\underline{y}) = +3.4$. Pro další bit v pořadí u_6 nám update sekvence značí, že všechny slučované cesty od pozice $k = 6$ až do konce

mřížového diagramu dávají pro tento bit rozdílnou hodnotu jak ML cesta. Nejnižší z metrických rozdílů dává opět cesta slučující se v $k = 6$ $(3, 4)$, a proto bude mít $L(u_6|\underline{y})$ opět hodnotu $+3, 4$. Toto dokazuje, jak může jedna zrušená cesta, která má malý rozdíl metrik, ovlivnit LLR pro všechny bity, které dávají rozdílné hodnoty od ML cesty. Algoritmus zvolil v $k = 6$ nenulovou cestu jako přežívající. Rozdíl metrik mezi touto přežívající (špatně zvolenou) a zrušenou cestou je $3, 4$. Z tohoto důvodu je hodnota LLR pro tyto tři špatně dekódované bity u_4, u_5 a u_6 nejnižší ze všech dekódovaných bitů. Zbývající hodnoty LLR se vypočítají podobným způsobem.

Pozice v MD k	Výstup dekodéru LLR $L(u_k\|\underline{y})$	A-priori info. $L(u_k)$	Přijatá sys. posl. $L_c y_{ks}$	Vnější informace
1	–12,4	0	–3,8	–8,6
2	–4,2	0	–1,8	–2,4
3	–8,8	0	–4,2	–4,6
4	+3,4	0	+2,8	+0,6
5	+3,4	0	+3,2	+0,2
6	+3,4	0	–2,6	+6
7	–3,8	0	–1	–4,8
8	–3,8	0	–5,4	–1,6
9	–4,8	0	–2,4	–2,4

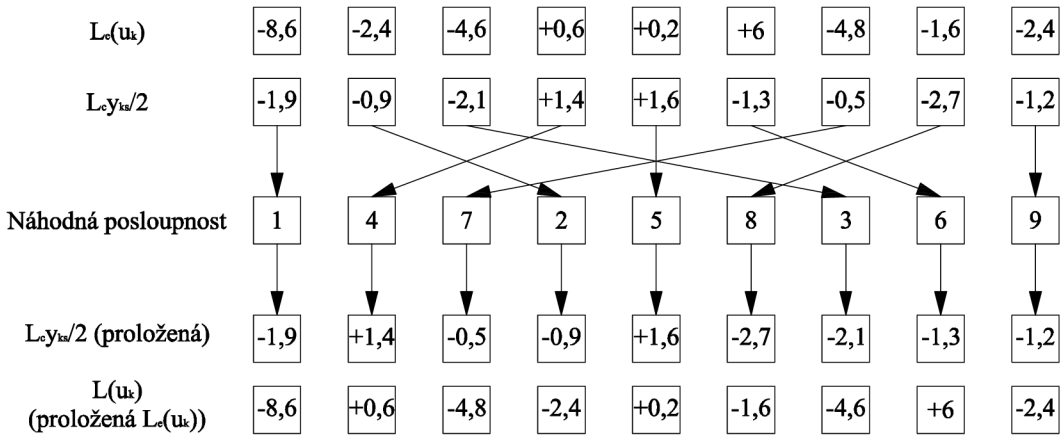
Tab. 1.3: Výpočet vnější informace z prvního dekodéru v první iteraci.

Nyní se přesuneme k popisu operací druhého dekodéru v první iteraci. Tento dekodér používá vnější informaci z prvního dekodéru jako a-priori informaci k tomu, aby napomáhala jeho výkonu, a proto by měl být schopen dát lepší odhad zakódované posloupnosti oproti prvnímu dekodéru. Vnější informace se vypočítá podle vzorce 1.11, který je zde pro přehlednost zopakován [5]:

$$L_e(u_k) = L(u_k \mid \underline{y}) - L(u_k) - L_c y_{ks}. \tag{1.20}$$

Tento vzorec dává vnější informaci $L_e(u_k)$, danou měkkým výstupem $L(u_k \mid \underline{y})$, z dekodéru od kterého se odečte a-priori informace $L(u_k)$ (jestliže je nějaká k dispozici) a přijatá systematická zpráva z přenosového kanálu $L_c y_{ks}$. V tab. 1.3 je znázorněna vnější informace z prvního dekodéru, vypočítaná pomocí vzorce 1.20, která je následně proložena náhodným prokladačem obr. 1.14 a použita jako a-priori informace

pro druhý dekodér. Druhý dekodér použije proložené (viz. obr. 1.14) přijaté systematické bity a paritní bity z druhého kodéru, které nebyly zúženy tj. druhý, čtvrtý, šestý a osmý bit.

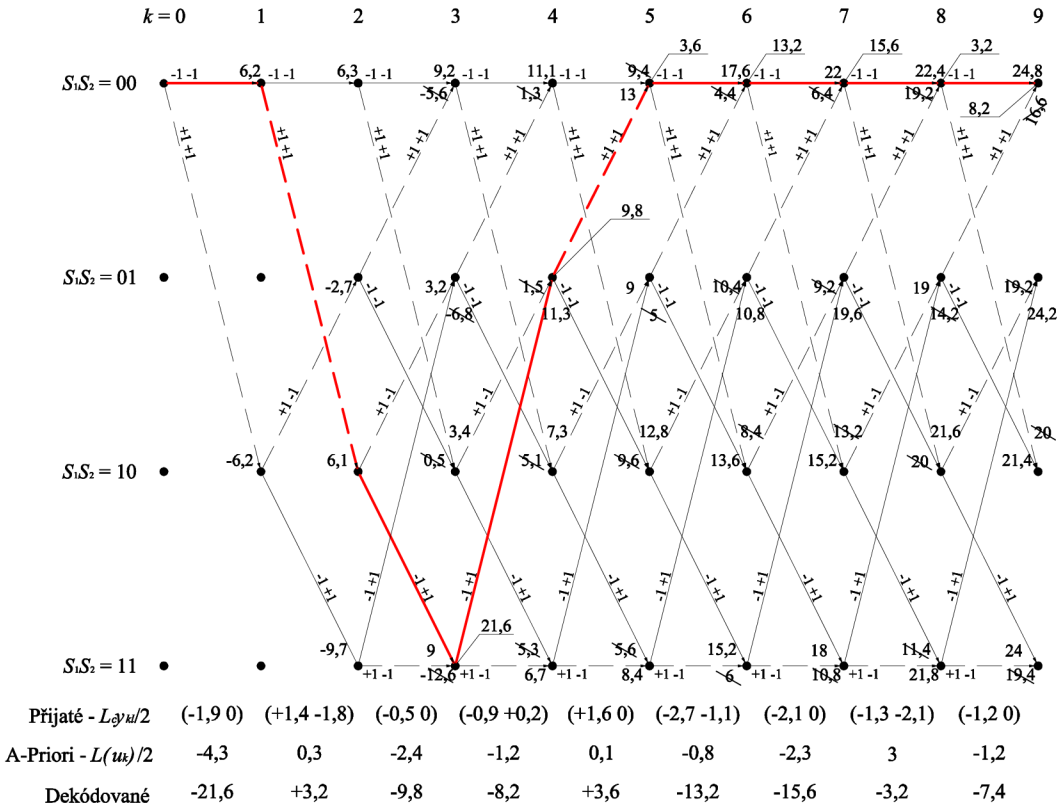


Obr. 1.14: Princip prokladače mezi prvním a druhým dekodérem.

Obr. 1.15 nám znázorňuje mřížový diagram pro SOVA dekódování druhého dekodéru v první iteraci, hodnoty proložené vnitřní informace $L(u_k)/2$ a přijaté posloupnosti z přenosového kanálu $L_c y_{kl}/2$. Pro druhý dekodér není mřížový diagram ukončen, cesty končí ve všech čtyřech možných stavech mřížového diagramu a jsou zvažovány dekodérem. Nicméně metrika pro stav $S_1S_2 = 00$ je maximální z těchto čtyř finálních stavů, a proto bude tento nulový stav považován za finální stav mřížového diagramu. Dále si všimněte, že metrika v obr. 1.15 se nyní vypočítává jako vzájemná korelace (cross corelation) přijaté a očekávané kanálové zprávy plus a-priori informace $L(u_k)/2$.

Vybraná ML cesta je znázorněna tučnou čarou společně s výstupními LLR hodnotami dekodéru. Ty jsou vypočítány za pomocí update sekvencí a minimálních metrických rozdílů (tab. 1.4) stejným způsobem, jako tomu bylo u prvního dekodéru obr. 1.13, tab. 1.2. Můžeme tedy vidět, že dekodér provedl nesprávné rozhodnutí v pozici $k = 5$ v mřížovém diagramu a zvolil nenulovou cestu jako cestu přežívající.

Nicméně chybně vybraná cesta dává nesprávné hodnoty dekódovaných bitů (+1) pouze pro dva přechody. Hodnoty dekódovaného LLR a uptade sekvence pro druhý



Obr. 1.15: Mřížový diagram SOVA dekódování pro druhý dekodér v první iteraci.

Pozice v MD k	Dekódovaný bit u_k	Rozdíl metrik Δ_k^s	Update sekvence	Výstup dekodéru LLR $L(u_k\|\underline{y})$
1	−1	–	–	−21,6
2	+1	–	–	+3,2
3	−1	21,6	111	−9,8
4	−1	9,8	1110	−8,2
5	+1	3,6	10010	+3,6
6	−1	13,2	101010	−13,2
7	−1	15,6	1001000	−15,6
8	−1	3,2	10000010	−3,2
9	−1	8,2	110001000	−7,4

Tab. 1.4: Výstup SOVA pro druhý dekodér v první iteraci.

dekodér jsou znázorněny v tabulce. LLR $L(u_k|\underline{y})$ pro bity nesprávně dekódované u_2 a u_5 jsou $+3,2$ a $+3,6$. Ty jsou výrazně nižší než ostatní LLR dekódovaných bitů, což signalizuje, že si v těchto dvou bitech není algoritmus jistý.

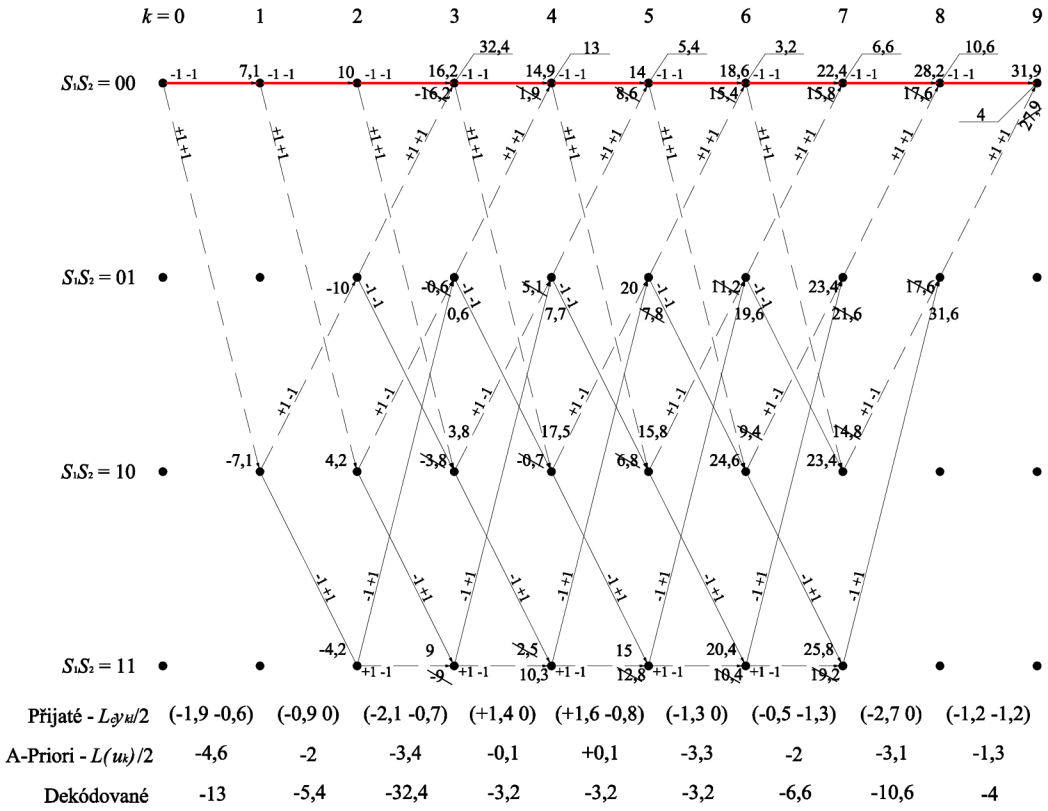
Když druhý dekodér vypočítá hodnoty LLR, dokončí tím první iteraci. Tyto

hodnoty jsou znázorněny ve spodním řádku na obr. 1.15. Nyní by mohly být zpětně proloženy a použity jako výstup z turbo dekodéru. Toto zpětné proložení by mělo dát výstupní posloupnost, která bude mít všechny hodnoty LLR záporné kromě u_4 a u_5, které by byly nesprávně dekódované jako +1. Tedy po jedné iteraci udělal turbo dekodér o jednu chybu méně, než konvoluční dekodér. Lepších výsledků dosáhneme, když použijeme více iterací, a tak tedy pokročíme k popisu operací v druhé iteraci.

Pozice v MD k	Výstup dekodéru LLR $L(u_k\|\underline{y})$	A-priori info. $L(u_k)$	Přijatá sys. posl. $L_c y_{ks}$	Vnější informace
1	–21,6	–8,6	–3,8	–9,2
2	+3,2	+0,6	–2,8	–0,2
3	–9,8	–4,8	–1	–4
4	–8,2	–2,4	–1,8	–4
5	+3,6	+0,2	+3,2	+0,2
6	–13,2	–1,6	–5,4	–6,2
7	–15,6	–4,6	–4,2	–6,8
8	–3,2	+6	–2,6	–6,6
9	–7,4	–2,4	–2,4	–2,6

Tab. 1.5: Výpočet vnější informace z druhého dekodéru v první iteraci.

V druhé iteraci první dekodér použije vnější informaci z druhého dekodéru z předchozí (první) iterace jako a-priori informaci. Tab. 1.5 znázorňuje vypočítané vnější informace z druhého dekodéru v první iteraci. Je zde vidět, že LLR pro všechny bity je záporné mimo bity u_2 a u_5, u kterých se LLR blíží nule. Tato vnější informace je zpětně proložena a použita jako a-priori informace pro první dekodér v druhé iteraci. Mřížový diagram pro tento dekodér je znázorněn na obr. 1.16 a je zřejmé, že dekodér použije stejnou zprávu z přenosového kanálu jako v první iteraci. Na rozdíl od obr. 1.12 má dekodér k dispozici a-priori informaci, která má napomáhat k nalezení správné cesty v mřížovém diagramu. Zvolená ML cesta je znovu znázorněna tlustou červenou čarou a také můžeme vidět, že byla zvolena správná cesta pro výstupní nulové hodnoty. Druhá iterace je následně dokončena uložením update sekvencí, vypočítáním LLR pro dekódované bity tab. 1.6 a nalezením vnější informace z prvního dekodéru, která je následně proložena a použita jako a-priori (vnitřní) informace pro druhý dekodér. Tento dekodér zvolí nulovou cestu jako ML cestu, a proto bude



Obr. 1.16: Mřížový diagram SOVA dekódování pro první dekodér v druhé iteraci.

výstup druhé iterace posloupnost nulových hodnot.

Pozice v MD k	Dekódovaný bit u_k	Rozdíl metrik Δ_k^s	Update sekvence	Výstup dekodéru LLR $L(u_k\|\underline{y})$
1	–1	–	–	–13
2	–1	–	–	–5,4
3	–1	32,4	111	–32,4
4	–1	13	1001	–3,2
5	–1	5,4	10010	–3,2
6	–1	3,2	111000	–3,2
7	–1	6,6	1001000	–6,6
8	–1	10,6	11011000	–10,6
9	–1	4	100011000	–4

Tab. 1.6: Výstup SOVA pro první dekodér v druhé iteraci.

2 TURBO BLOKOVÉ KÓDY

2.1 Úvod

Typicky používané kodéry pro turbo kódování jsou tak zvané RSC (rekurzivní systematický konvoluční – Recursive Systematic Convolutional). Jak ukázal Hagenauer, je možné použít i kodéry blokové, které jsou schopné pracovat na informační rychlosti blížící se jedné. Blokové kódy jsou více vhodné pro tuto informační rychlost, ale turbo blokové kódy se vyznačují vysokou dekódovací složitostí. Z tohoto důvodu se pro poměr nižší jak 2/3 používají turbo kódy s konvolučními kodéry. V této kapitole se budeme věnovat běžné struktuře turbo kódů používající binární BCH (Bose-Chaudhuri-Hocquenghem) kódy [5].

Turbo blokové kódy [4] [5] je možné dekódovat pomocí algebraického dekódování nebo pomocí mřížového dekódování, které bude tématem této kapitoly. Příkladem mřížového dekódování je SOVA (měkký výstup Viterbiho algoritmu – Soft-Output Viterbi Algorithm) nebo MAP (Maximum A-Posteriori) algoritmus [3] [4] [5] [7] [10].

2.2 BCH kódování

BCH kodér přijímá jako vstupní posloupnost k vstupních bitů a produkuje n výstupních bitů. Minimální Hammingova vzdálenost kódového slova je d_{min}. BCH kód je pak značený jako BCH (n, k, d_{min}). V tab. 2.1 [5] nalezneme často používané vytvářecí mnohočleny $g(x)$. Koeficient $g(x)$ je zadán v osmičkové soustavě a je nutné ho převést do soustavy binární.

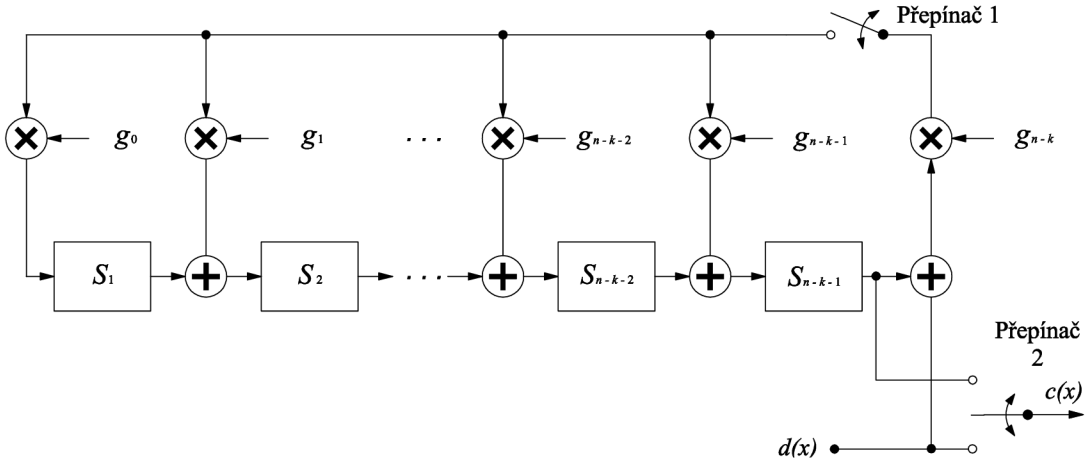
Protože BCH kódy jsou cyklické kódy, je možné je implementovat použitím posuvných registrů. Kódy mohou kódovat buď nesystematicky, nebo systematicky. V této kapitole budeme mluvit pouze o systematických kódech. Pro systematické kódy se vytvářecí mnohočlen $g(x)$ zapsal jako:

$$g(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{n-k-1} x^{n-k-1} + g_{n-k} x^{n-k}. \tag{2.1}$$

n	k	d_{min}	$g(x)$ vytvářecí mnohočlen (osmičková soustava)
7	4	3	13
15	11	3	23
	7	5	721
31	26	3	45
	21	5	3551
	16	7	107657
63	57	3	103
	51	5	12471
	45	7	1701317
	39	9	166623567
	36	11	1033500423
127	120	3	211
	⋮	⋮	⋮
	64	21	1206534025570773100045
255	247	3	435
	⋮	⋮	⋮
	131	37	215713331471510151261250277442142024165471

Tab. 2.1: Tabulka BCH kódů.

Vytvářecí mnohočlen $g(x)$ určuje n bitů kódového slova, a to tak, že přidá ke k informačním bitům $(n - k)$ paritních bitů. Kodér používá $(n - k)$ posuvných registrů obr. 2.1, kde $\otimes$ je násobení a $\oplus$ je modulo - 2 (XOR). Paritní bity se vypočítají z bitů informačních dat podle pravidel daných vytvářecím mnohočlenem.



Obr. 2.1: Systematický kodér pro BCH (7, 4, 3) s $(n - k)$ posuvnými registry.

Následující kroky popisují proces kódování:

1. Přepínač 1 je sepnutý po dobu prvních k posunů. Informační bity $d(x)$ se posouvají v pořadí na posuvné registry.

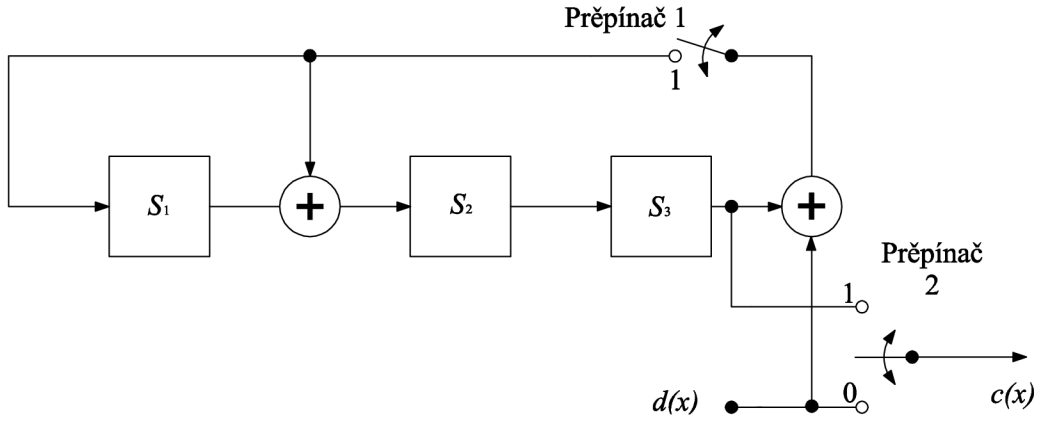
2. Přepínač 2 je ve spodní pozici a informační data $d(x)$ se v podstatě zkopírují do výstupu $c(x)$.

3. Po k-tém posunu se přepínač 1 rozepne a přepínač 2 přepne do horní polohy.

4. Následně se vyčistí posuvné registry, a to tak, že jejich aktuální hodnoty se použijí jako paritní bity.

Na příkladu si ukážeme, jakým způsobem pracuje kódování, a to na kódu BCH (7, 4, 3) daný vytvářecím mnohočlenem tab. 2.1:

$$\begin{aligned} g &= 13_{octal} \\ &= 1011_{bin} \\ g(x) &= x^3 + x + 1. \end{aligned} \tag{2.2}$$



Obr. 2.2: Systematický kodér pro BCH (7, 4, 3) s $(n - k) = 3$ posuvnými registry.

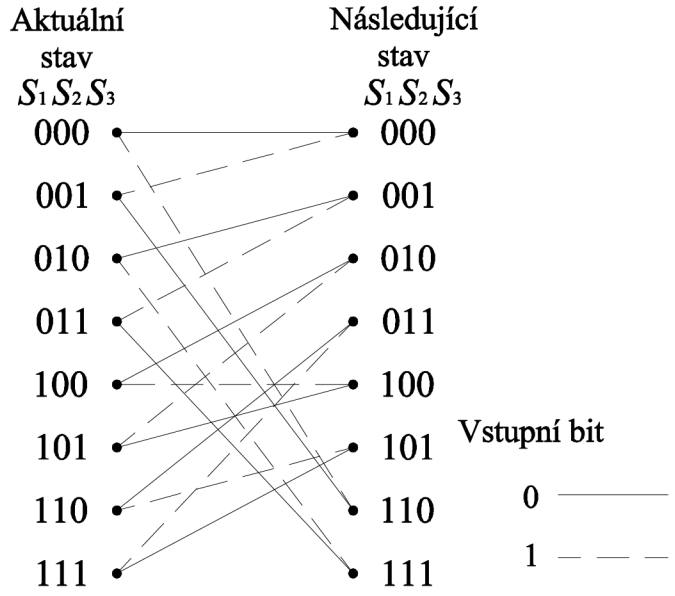
Na obr. 2.2 je ukázán kodér pro tento kód, který je odvozený z obr. 2.1. Všimněte si, že na obr. 2.2 nejsou znázorněny bloky násobení. V případě, že ve vytvářecím mnohočlenu je 1, tak se násobení zamění pouze za cestu a v případě 0 je násobení odstraněno i s cestou.

Nyní použijeme posuvné registry na obr. 2.2 k zakódování čtyř informačních bitů. Jednotlivé kroky jsou znázorněny v tab. 2.2.

Vstupní posloupnost	Krok	Posuvný registr $S_1S_2S_3$	Kódové slovo $c_1c_2c_3c_4c_5c_6c_7$	Přep. 1	Přep. 2
1 0 1 1	1	0 0 0	– – – – – – –	1	0
1 0 1	2	1 1 0	– – – – – – 1	1	0
1 0	3	1 0 1	– – – – – 1 1	1	0
1	4	1 0 0	– – – – 0 1 1	1	0
–	5	1 0 0	– – – 1 0 1 1	1	0
–	6	0 1 0	– – 0 1 0 1 1	0	1
–	7	0 0 1	– 0 0 1 0 1 1	0	1
–	8	0 0 0	1 0 0 1 0 1 1	0	1

Tab. 2.2: Tabulka vstupních a výstupních stavů BCH kodéru.

Posuvné registry musí být před kódováním vynulovány. Po čtyřech krocích se přepínač 1 rozepne a přepínač 2 přepne do horní polohy.

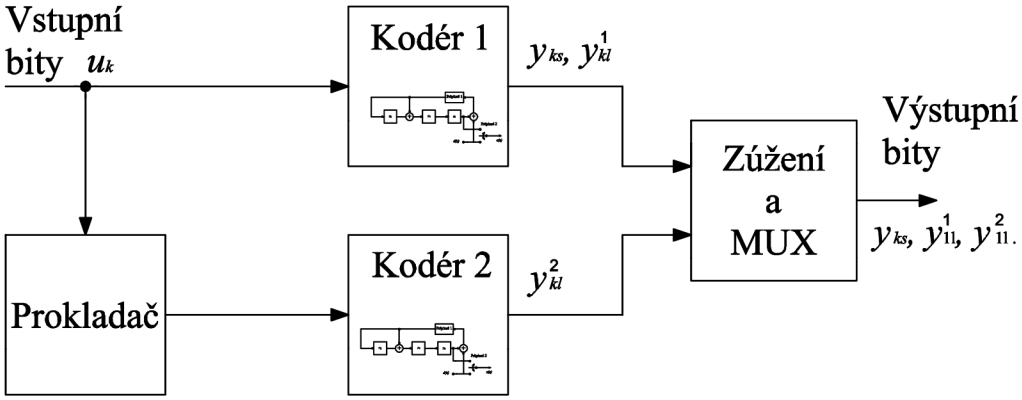


Obr. 2.3: Mřížový diagram pro BCH (7, 4, 3) kodér.

Paritní bity na posuvných registrech jsou přidány na výstup. Kódové slovo je $c = 1$ 0 0 1 0 1 1 ($c(x) = 1 + x^3 + x^5 + x^6$). Mřížový diagram pro BCH (7, 4, 3) kodéru je znázorněn na obr. 2.3.

2.3 Turbo BCH kódování

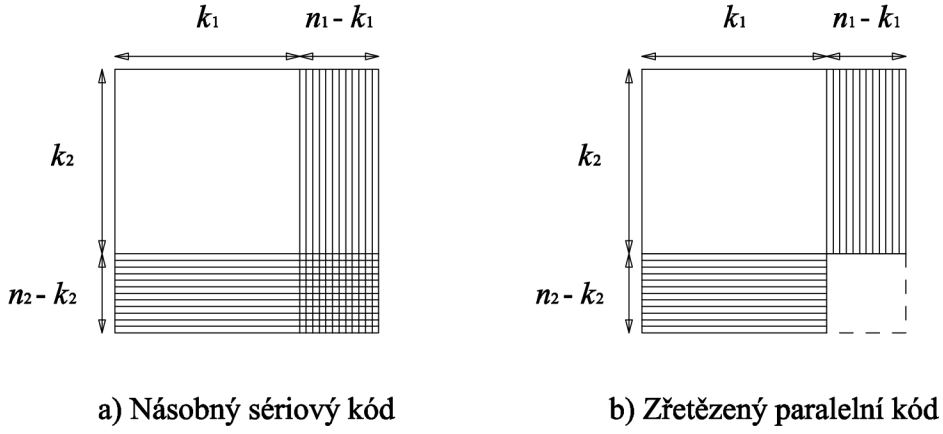
Základní struktura turbo BCH kodéru je zobrazena na obr. 2.4. Jsou zde použity dva BCH kodéry, mezi které je vložen prokladač kap. 1.4. Je zde možné použít blokové nebo náhodné prokládání.



Obr. 2.4: Blokové schéma turbo BCH kodéru.

Vzhledem ke struktuře turbo kodéru zobrazeného na obr. 2.4, je často nazýván jako zřetězený paralelní kód. Zřetězené paralelní kódy tvoří takzvaný násobný sériový kód. Obecně se násobný kód skládá ze dvou lineárních kódů C_1 a C_2, kde C_1 a C_2 mají parametry (n_1, k_1, d_{min1}) a (n_2, k_2, d_{min2}). Většinou se $C_1 = C_2$. Násobný kód je získán vložením bitů informačních dat $(k_1 \times k_2)$ do pole o k_1 sloupců a k_2 řádků. Data v k_1 sloupcích a k_2 řádcích jsou kódovány kodéry C_1 a C_2. $(n_1 - k_1)$ posledních sloupců je kódové slovo kodéru C_2 a přesně $(n_2 - k_2)$ posledních řádků je kódové slovo kodéru C_1. Mimo to, parametry vyplývající z násobných kódů, jsou $n = n_1 \times n_2$, $k = k_1 \times k_2$ a $d_{min} = d_{min1} \times d_{min2}$, zatímco poměr kódu je $\frac{k_1}{n_1} \times \frac{k_2}{n_2}$. Struktura zřetězených paralelních kódů je stejná jako u násobných kódů, kromě toho, že nadbytečná část vyplývající z kontrolování parit z paritní části obou kodérů C_1 a C_2 se vynechává. Hlavní nevýhodou zřetězených paralelních kódů je ztráta minimální

kódové vzdálenosti, která je $d_{min1} + d_{min2} - 1$, ve srovnání s $d_{min1} \times d_{min2}$ násobných kódů.



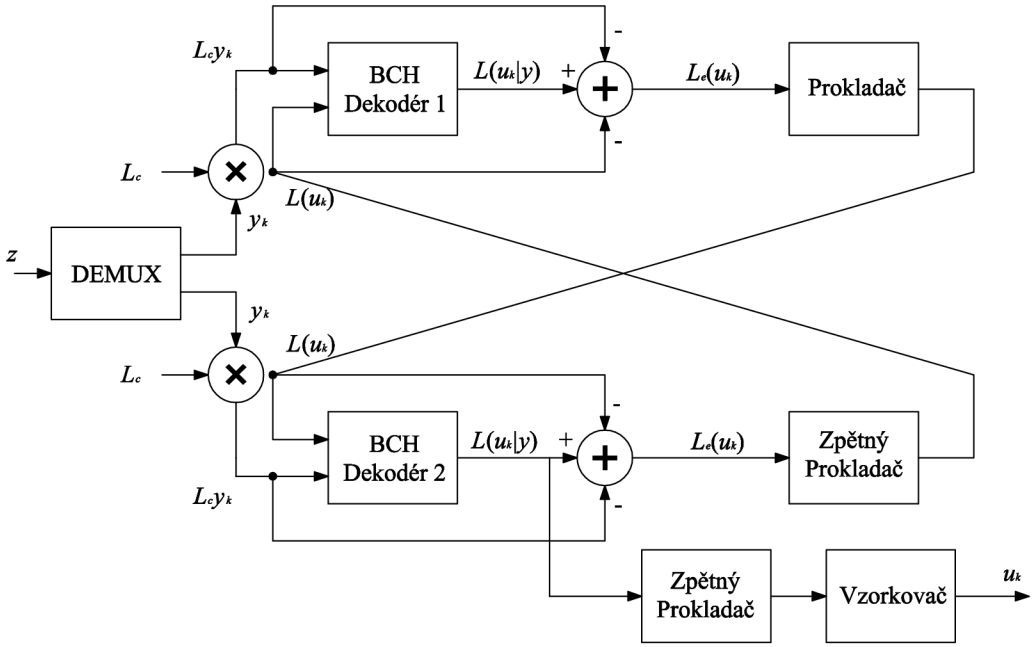
Obr. 2.5: Konstrukce násobných sériových a zřetězených paralelních kódů.

Výstupní bity z obou BCH kodérů jsou zúženy (děrovány) a následně multiplexovány. V tab. 2.1 je znázorněn rozsah BCH kódů s různými informačními rychlostmi.

2.4 Turbo BCH dekódování

Na obr. 2.6 je zobrazena struktura turbo blokového dekodéru. Jednotlivé vstupy a výstupy jsou popsány následovně z jsou přijaté vzorky, u_k dekódované bity, $L_c y_k$ demultiplexovaný měkký výstup, $L(u_k)$ vnitřní informace (a-priori informace), $L_e(u_k)$ vnější informace a $L(u_k \mid \underline{y})$ a-posteriori informace. Bližší informace k tomuto značení najdeme v kapitolách 1.6.1 a 1.6.3. Z obrázku je zřejmé, že turbo dekodér používá dva BCH dekodéry. Vzhledem k tomu, že použijeme metodu mřížového dekódování, je tedy možné použít algoritmy jako je SOVA (měkký výstup Viterbiho algoritmu – Soft-Output Viterbi Algorithm) [3] [4] [5] nebo MAP (Maximum A-Posteriori) [3] [4] [5].

Dekodér používá měkký kanálový výstup $L_c y$ a vnitřní informaci $L(u_k)$ k tomu, aby mohl následně poskytnout na výstupu a-posteriori informaci $L(u_k \mid \underline{y})$. Vnější informace $L_e(u_k)$ je dána odečtením měkkého výstupu kanálu $L_c y$ a vnitřní informace $L(u_k)$ od a-posterioni informace $L(u_k \mid \underline{y})$ podle:



Obr. 2.6: Schéma turbo BCH dekodéru.

$$L_e(u_k) = L(u_k \mid \underline{y}) - L_c y_k - L(u_k). \tag{2.3}$$

Poté, co je podle obr. 2.6 proložena nebo zpětně proložena, se vnější informace $L_e(u_k)$ stává vnitřní informací pro druhý dekodér. Stejným způsobem se vnější informace z druhého dekodéru vrací zpět do prvního dekodéru jako vnitřní informace. Oba dekodéry si vzájemně napomáhají vyměňováním informací souvisejících s datovými bity, proto se jedná o iterativní dekódování.

2.4.1 Matematický popis SOVA (BCH)

Matematický popis SOVA pro dekódování BCH kódů je velmi podobný matematickému popisu pro RSC kódy, který byl popsán v kapitole 1.6.3. Jak už bylo řečeno SOVA má pro turbo dekódování dvě modifikace.

První modifikace říká, že metrika cesty je modifikována tak, aby počítala s vnitřní informací $L(u_k)$. Druhá modifikace pro algoritmus je poskytování a-posteriori informace $L(u_k \mid \underline{y})$ pro každý dekódovaný bit.

Kódové slovo BCH kódu, je složeno z datových a paritních bitů. V turbo blokovém dekódování projde přes dekodér pouze měkký výstup původních datových bitů do dalšího dekodéru. Není zde vnitřní informace $L(u_k)$ pro paritní bity. Výpočet metriky cesty je tedy [5]:

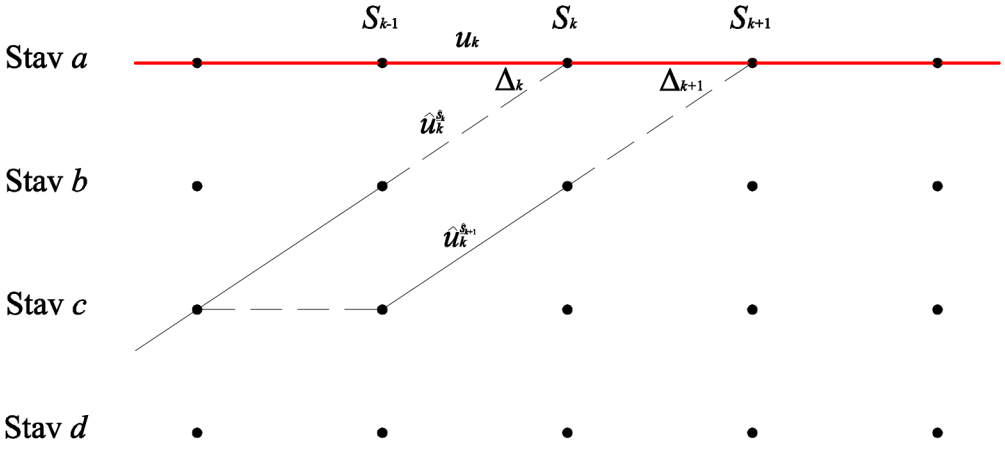
$$M(\underline{s}_k) = M(\underline{s}_{k-1}) + \frac{u_k}{2}\left\{L(u_k) + L_c y_k\right\}. \tag{2.4}$$

Nyní se budeme věnovat výše uvedené druhé modifikaci Viterbiho algoritmu, a to a-posteriori informaci $L(u_k \mid \underline{y})$, pomocí které počítáme vnější informaci $L_e(u_k)$. V mřížovém diagramu, kde dvě cesty dosahují stavu $S_k = s$, bude cesta s nižší metrikou vyřazena. Ponechaná cesta se značí $\underline{s}_k$ a vymazaná $\hat{\underline{s}}_k$. Rozdíl metrik těchto cest se vypočítá jako [5]:

$$\Delta_k = M(\underline{s}_k) - M(\hat{\underline{s}}_k) \geq 0. \tag{2.5}$$

LLR rozhodnutí ML cesty $\underline{s}_k$ se vypočítá:

$$L\{P(\underline{s}_k)\} = \ln \frac{P(\underline{s}_k)}{1 - P(\underline{s}_k)} = \Delta_k. \tag{2.6}$$



Obr. 2.7: Zjednodušený mřížový diagram pro BCH dekódování.

Nyní potřebujeme získat LLR dekódovaných bitů. Na obr. 2.7 je znázorněn zjednodušený mřížový diagram, kde ML cesta představuje posloupnost nul. Z obrázku je zřejmé, že cesta $c \longrightarrow b \longrightarrow a$, tedy cesta $\hat{s}_k$ a cesta $c \longrightarrow c \longrightarrow b \longrightarrow a$ $\underline{\hat{s}}_{k+1}$ jsou cesty zrušené v pozici k a $k+1$. Tedy $\hat{u}_k^{\hat{\underline{s}}_k}$ a $\hat{u}_k^{\hat{\underline{s}}_{k+1}}$ jsou dekódované bity ve stavu k, které by byly výstupem pro vyřazené cesty $\underline{\hat{s}}_k$ a $\underline{\hat{s}}_{k+1}$ v případě, že by tyto cesty nebyly vyřazeny. Jak je vidět na obrázku u_k a $\hat{u}_k^{\hat{\underline{s}}_k}$ jsou odlišné. Z tohoto důvodu je LLR aktuálně dekódovaného bitu u_k rovno Δ_k. Naopak pro bit u_k a $\hat{u}_k^{\hat{\underline{s}}_{k+1}}$ by nedošlo k chybě, protože dekódovaná hodnota bitu je stejná bez ohledu na to, jestli by vybraná cesta byla s_k nebo $\underline{\hat{s}}_{k+1}$. Pak by byla hodnota LLR pro bit u_k ∞. Vztah pro LLR bitu u_k, beroucí v úvahu zrušenou cestu $\underline{\hat{s}}_{k+i}$ je definován jako [5]:

$$L_{\underline{\hat{s}}_{k+i}}(u_k) = u_k \cdot \begin{cases} \infty & \text{když} \quad u_k = \hat{u}_k^{\hat{\underline{s}}_{k+i}} \\ \Delta_k & \text{když} \quad u_k \neq \hat{u}_k^{\hat{\underline{s}}_{k+i}}, \end{cases} \tag{2.7}$$

kde $i \geq 0$. Nicméně vztah 2.7 pracuje pouze s jednou zrušenou cestou, zatímco podél ML cesty může být více zrušených cest. Je tedy výhodnější počítat LLR $L(u_k \mid \underline{y})$ podle následujícího vzorce [5]:

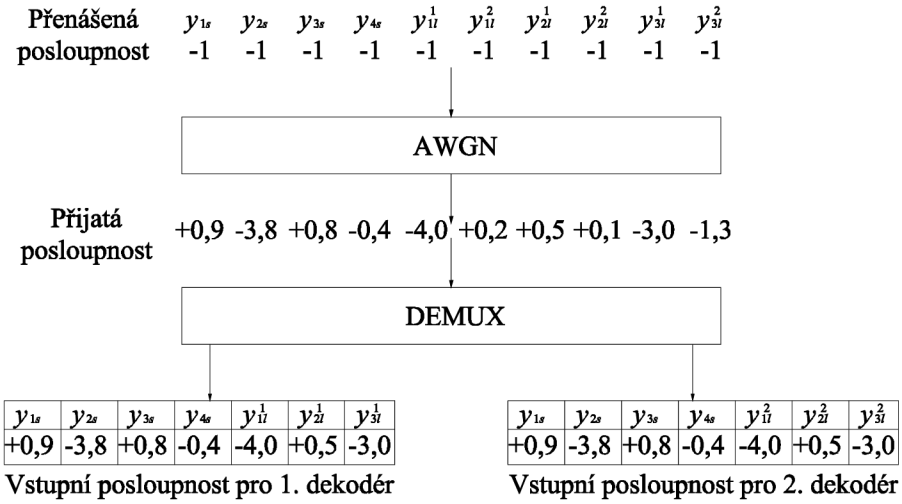
$$L(u_k \mid \underline{y}) = u_k \cdot \min_{j=k\cdots n \;\; u_k \neq \hat{u}_k^{\hat{\underline{s}}_j}} \Delta_j. \tag{2.8}$$

Algoritmus zkoumá, jestli nedošlo ke špatnému rozhodnutí v mřížovém diagramu. Tyto potenciálně nespolehlivé rozhodnutí jsou spojené s vyřazením cesty s podobnou metrikou, jakou měla cesta přežívající, a tedy se tyto cesty vyznačují malými rozdíly mezi metrikami Δ. Z tohoto důvodu hledá algoritmus uzly, kde byl tento rozdíl nejmenší a prověřuje tyto vyřazené cesty, aby nedošlo ke špatnému dekódování bitu.

2.5 Příklad turbo blokového dekódování pomocí SOVA

Tato kapitola se věnuje turbo blokovému dekódování, které používá SOVA [4][5]. V tomto příkladu si ukážeme, jak iterativní dekódování napomáhá při korekci více-

násobných chyb. Kodéry použité v tomto příkladu používají BCH (7,4,3) kód a jsou kombinované s náhodným prokladačem obr. 2.4. Jak můžeme vidět na zmíněném obrázku, paritní bity p_k^1 generované prvním BCH kodérem a paritní bity p_k^2 z druhého BCH kodéru nejsou zúženy. Vzhledem k tomu, že výstupní datové bity jsou stejné pro oba kodéry, budou tyto bity z druhého kodéru zúženy. Z tohoto důvodu bude přenesená posloupnost obsahovat čtyři datové a šest paritních bitů. Pro jednoduchost použijeme všechny vstupní bity s nulovou hodnotou, proto bude výstupní posloupnost z obou kodérů nulová. Pro přenos použijeme BPSK modulaci a z tohoto důvodu budeme logickou 0 přenášet jako -1 a logickou 1 jako $+1$. Přenášená posloupnost bude tedy tvořena deseti -1.



Obr. 2.8: Demultiplexování přijaté posloupnosti.

Z obr. 2.8 je zřejmé, že přenášená posloupnost, která má energii $E_b = 1$ na bit, je přenášena přes AWGN kanál. Protože v kanále nedochází ke kolísání, amplituda $a = 1$ a rozptyl $\sigma = \sqrt{2}$. Proto bude hodnota spolehlivosti přenosového kanálu L_c daná vzorcem 1.10[5]:

$$\begin{aligned} L_c &= 4a\frac{E_b}{2\sigma^2} \\ &= 1. \end{aligned} \tag{2.9}$$

Přijatá posloupnost $\underline{y}$ je zarušena šumem, přivedena do demultiplexoru, který jí rozdělí na vstupní posloupnosti pro první a druhý dekodér obr. 2.8. Hodnoty přijatých vzorků originálních datových bitů jsou pro oba dekodéry stejné, ale pro druhý dekodér je nutné tyto vzorky proložit. Proložením zvyšujeme korekční schopnost dekodéru, a to proto, že když první dekodér nedokáže opravit chybu, druhý dekodér použije přenesenou posloupnost přijatých datových bitů, aby opravil chyby, které neopravil předchozí dekodér a naopak. V následující části si ukážeme, jakým způsobem si dekodéry navzájem napomáhají v opravování chyb.

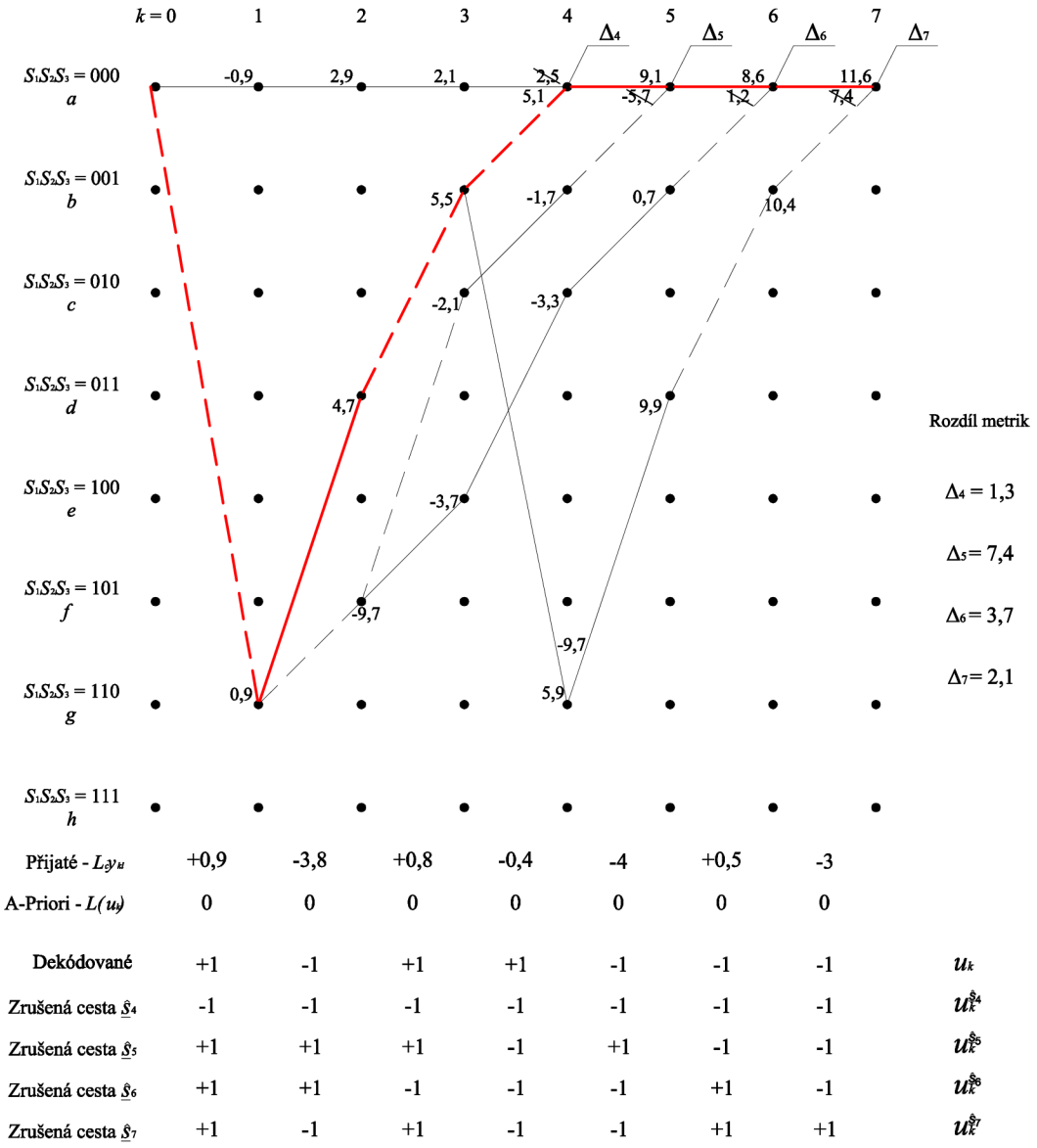
Metrika cesty se pro SOVA vypočítá pomocí vzorce 2.4, který je zde pro jednoduchost zopakován [5]:

$$M(\underline{s}_k) = M(\underline{s}_{k-1}) + \frac{u_k}{2}\left\{L(u_k) + L_c y_k\right\}, \tag{2.10}$$

kde $M(\underline{s}_{k-1})$ je metrika přežívající cesty v $S_{k-1} = \grave{s}$ v pozici $k-1$ v mřížovém diagramu, u_k je předpokládaný přenesený bit spojený s daným přechodem, y_k je hodnota výstupu z přenosového kanálu pro tento přechod a L_c je spolehlivost přenosového kanálu. Proto $L_c y_k = y_k$.

Nyní se budeme věnovat prvnímu dekodéru v první iteraci, kterému odpovídá mřížový diagram na obr. 2.9. Pro první dekodér v první iteraci není a-priori informace, a proto je $L(u_k) = 0$ pro všechny k. Přijatý signál $L_c y_k$ je vstupní posloupností pro první dekodér obr. 2.8. Podle vzorce 2.10 je metrika přechodu daná součtem $L(u_k) = 0$ a přijatým signálem $L_c y_k$. Tyto hodnoty jsou znázorněny pod mřížovým diagramem v obr. 2.9.

SOVA postupuje stejným způsobem jako Viterbiho algoritmus, a to tak, že hledá ML cestu, která je znázorněna v obr. 2.9 tlustou červenou čarou a je tedy zřejmé, že ML cesta prvního dekodéru v první iteraci prochází stavy $a \to g \to d \to b \to a \to a \to a \to a$, která nedává posloupnost nul. Dále jsou zde znázorněny zrušené cesty podél ML cesty. Dekódované bity pro ML cestu u_k a pro cesty zrušené $u_k^{\hat{\underline{s}}_k}$ jsou znázorněny ve spodní části obr. 2.9. Pomocí následujícího vzorce vypočítáme rozdíl metrik cest [5]:



Obr. 2.9: Mřížový diagram SOVA dekódování pro první dekodér v první iteraci (BCH).

$$\Delta_k = M(\underline{s}_k) - M(\underline{\hat{s}}_k), \tag{2.11}$$

kde $M(\underline{s}_k)$ je metrika ML cesty $\underline{s}$ a $M(\underline{\hat{s}}_k)$ je metrika zrušené cesty $\underline{\hat{s}}$. V obr. 2.9 jsou znázorněny čtyři zrušené cesty, proto zde budou čtyři rozdíly v metrikách cest Δ_k. Tyto rozdíly jsou znázorněny vpravo od mřížového diagramu. Jako příklad vezmeme

pozici $k = 4$ ve stavu a. Zrušená cesta je zde $a \to a \to a \to a \to a$, tedy rozdíl metrik cest $\Delta_4 = (5, 1 - 2, 5)/2 = 1, 3$. Podobně je tomu v pozici $k = 5$ ve stavu a, kde zrušená cesta prochází stavy $a \to g \to f \to c \to b \to a$ a rozdíl metrik $\Delta_5 = (9, 1 - (-5, 7))/2 = 7, 4$. Následující rozdíly metrik se počítají stejným způsobem.

	$L_{\widehat{\underline{s}}_k}(u_k)$						
k	$\left\|L_{\widehat{\underline{s}}_4}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}}_5}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}}_6}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}}_7}(u_k)\right\|$	Min	u_k	$L(u_k\|\underline{y})$
1	1,3	∞	∞	∞	1,3	+1	+1,3
2	∞	7,4	3,7	∞	3,7	–1	–3,7
3	1,3	∞	3,7	∞	1,3	+1	+1,3
4	1,3	7,4	3,7	2,1	1,3	–1	+1,3

Tab. 2.3: Výstup SOVA pro první dekodér v první iteraci (BCH).

A-posteriori informace $\left|L_{\widehat{\underline{s}}_k}(u_k)\right|$ dekódovaného bitu u_k, znázorněná v tab. 2.3, bere v úvahu zrušenou cestu $\hat{\underline{s}}_k$. Podle vzorce 2.7 se $\left|L_{\widehat{\underline{s}}_k}(u_k)\right| = \Delta_k$, když u_k je rozdílné od $u_k^{\hat{\underline{s}}_k}$ jinak se $\left|L_{\widehat{\underline{s}}_k}(u_k)\right| = \infty$. Jestliže zrušená cesta vede částečně po ML cestě, považují se dekódované bity u_k a $u_k^{\hat{\underline{s}}_k}$ za stejné. Vysvětlení si uvedeme na příkladu. V obr. 2.9 se ML cesta s_4 sbíhá se zrušenou nulovou cestou $\hat{\underline{s}}_4$ v pozici $k = 4$ a rozdíl metrik cest je $\Delta_4 = 1, 3$. Dekódované bity ML a zrušené cesty se liší v pozicích $k = 1$, 3 a 4, proto bude hodnota a-posteriori informace $\left|L_{\widehat{\underline{s}}_4}(u_k)\right|$ pro dekódované bity u_k, $k = 1$, 3 a 4 rovna rozdílu metrik cest $\Delta_4 = 1, 3$. V případě $k = 2$ se bude a-posteriori informace $\left|L_{\widehat{\underline{s}}_4}(u_1)\right| = \infty$. Podobně je tomu v $k = 5$, kde dekódované datové bity pro ML cestu jsou +1, −1, +1, +1 a pro zrušenou cestu $\hat{\underline{s}}_5$ +1, +1, +1, −1. Když porovnáme tyto dvě posloupnosti, zjistíme, že se liší v pozici $k = 2$, 4 $\left(\left|L_{\widehat{\underline{s}}_5}(u_k)\right| = 7, 4\right)$, a že v pozicích $k = 1$, 3 jsou stejné $\left(\left|L_{\widehat{\underline{s}}_5}(u_k)\right| = \infty\right)$. Další hodnoty pro a-posteriori informaci jsou znázorněny v tab. 2.3. K odvození a-posteriori informace použijeme následující vzorec [5]:

$$L(u_k \mid \underline{y}) = u_k \cdot \min_{j=k\cdots n \;\; u_k \neq \hat{u}_k^{\hat{\underline{s}}_j}} \Delta_j. \tag{2.12}$$

V tab. 2.3 v pozic $k = 4$ jsou hodnoty $\left|L_{\widehat{\underline{s}}_4}(u_3)\right|$, $\left|L_{\widehat{\underline{s}}_5}(u_3)\right|$, $\left|L_{\widehat{\underline{s}}_6}(u_3)\right|$ a $\left|L_{\widehat{\underline{s}}_7}(u_3)\right|$ 1, 3, 7, 4, 3, 7, 2, 1, odpovídají čtyřem vyřazeným cestám a minimální rozdíl metrik je

1, 3. Použitím vzorce 2.12 můžeme odvodit a-posteriori informaci pro bit u_4, která je $L(u_4|\underline{y}) = +1,3$, protože dekódovaný bit $u_4 = +1$. Na obr. 2.6 je znázorněno jakým způsobem je dána vnější informace $L_e(u_k)$, která se vypočítá pomocí vzorce 2.3, který je pro přehlednost zopakován [5]:

$$L_e(u_k) = L(u_k \mid \underline{y}) - L_c y_k - L(u_k). \tag{2.13}$$

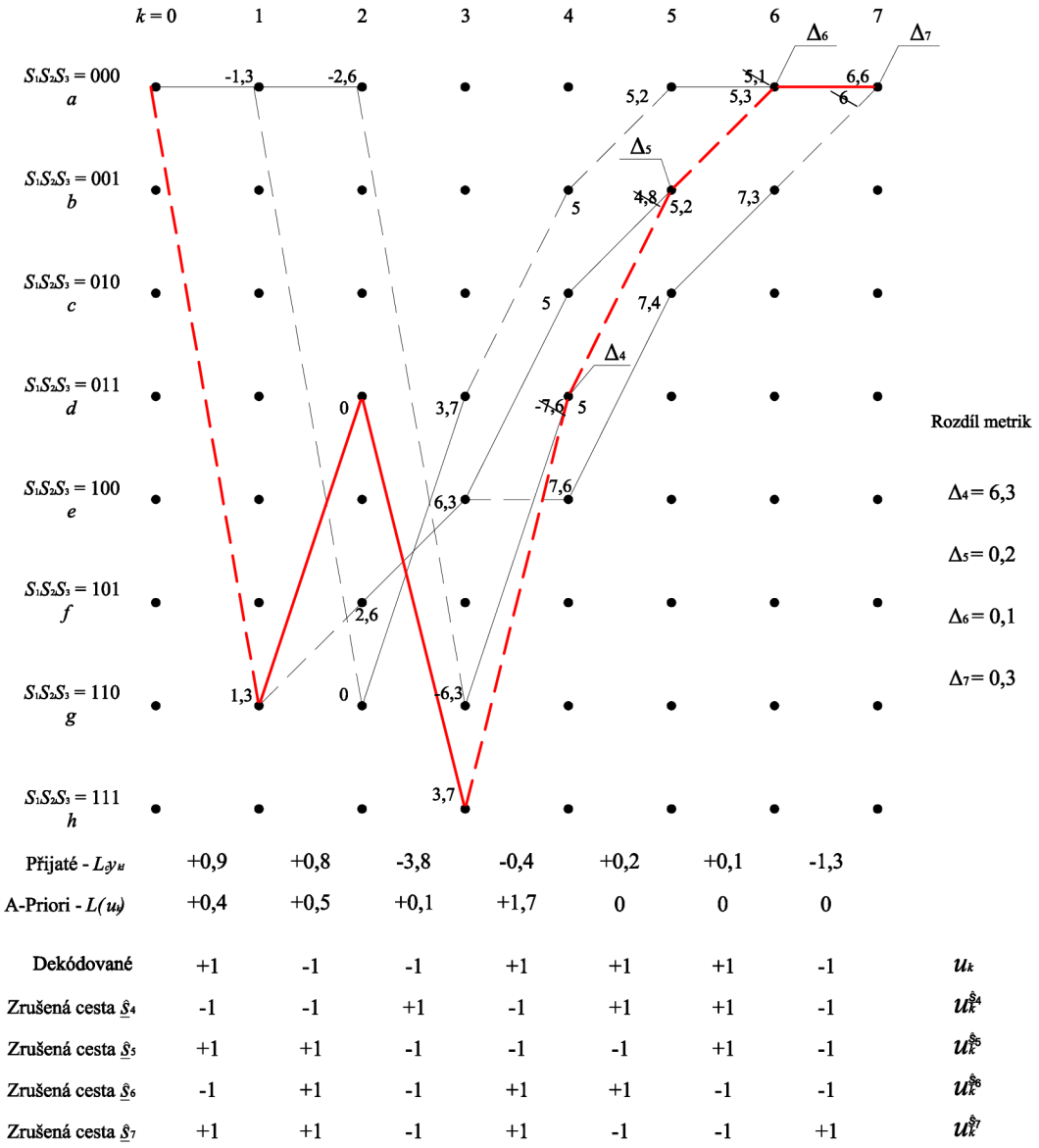
Ze vzorce je zřejmé, že vnější informace $L_e(u_k)$ je dána odečtením vnitřní informace $L(u_k)$ a přijatým signálem $L_c y_k$ od a-posteriori informace $L(u_k|\underline{y})$ z dekodéru. Poslední sloupec v tab. 2.4 znázorňuje vnější informaci vypočítanou pomocí vzorce 2.13.

k	$L(u_k\|\underline{y})$	$L_c y_k$	$L(u_k)$	$L_e(u_k)$
1	+1,3	+0,9	0	+0,4
2	–3,7	–3,8	0	+0,1
3	+1,3	+0,8	0	+0,5
4	+1,3	–0,4	0	+1,7

Tab. 2.4: Výpočet vnější informace z prvního dekodéru v první iteraci (BCH).

Nyní přejdeme k popisu operací druhého dekodéru v první iteraci. A-priori informace $L(u_k)$ pro druhý dekodér je v podstatě vnější informace produkovaná prvním dekodérem, která je proložena v prokladači. Dále jsou v prokladači proloženy vstupní vzorky datových bitů $L_c y_k$, které se připojí k paritním bitům z druhého BCH kodéru. Obr. 2.10 nám znázorňuje mřížový diagram pro druhý dekodér během první iterace. Vnější informace z tab. 2.4 je po proložení znázorněna pod mřížovým diagramem jako a-priori informace $L(u_k)$. Také je zde ukázaný přijatý signál $L_c y_k$. Dekódované bity pro ML a vyřazené cesty jsou znázorněny ve spodní části obrázku.

ML cesta zvolená druhým dekodérem je znázorněna jako tlustá červená čára. Tato cesta prochází stavy $a \to g \to d \to h \to d \to b \to a \to a \to$ a opět jako v předchozím případě se nejedná o cestu, pro kterou by byly dekódované bity posloupnost nul. Pomocí vzorce 2.11 vypočítáme metrické rozdíly Δ_k mezi ML cestou a cestami zrušenými. Tyto rozdíly v metrikách jsou znázorněny vpravo od mřížového diagramu v obr. 2.10. Všimněte si, že v obr. 2.9 byla vnitřní informace nula, zatímco



Obr. 2.10: Mřížový diagram SOVA dekódování pro druhý dekodér v první iteraci (BCH).

v obr. 2.10 máme hodnoty vnitřní informace $L(u_k)$ z prvního dekodéru. Tyto hodnoty podstatně mění metriku přechodu a metrické rozdíly cest.

Použitím stejných kroků, jako u prvního dekodéru obr. 2.9, můžeme vypočítat a-posteriori informaci $L(u_k|\underline{y})$ a vnější informaci $L_e(u_k)$, které jsou znázorněny v tab. 2.5 a tab. 2.6. Nyní si obr. 2.10 rozebereme více do hloubky. V kroku $k = 4$ ve stavu d je rozdíl metrik cest $\Delta_4 = 6,3$ mezi přežívající a zrušenou cestou. Jejich

k	$L_{\widehat{\underline{s}_k}}(u_k)$				Min	u_k	$L(u_k\|\underline{y})$
	$\left\|L_{\widehat{\underline{s}_4}}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}_5}}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}_6}}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}_7}}(u_k)\right\|$			
1	6,3	∞	0,1	∞	0,1	+1	+0,1
2	∞	0,2	0,1	0,3	0,1	–1	–0,1
3	6,3	∞	∞	∞	6,3	–1	–6,3
4	6,3	0,2	∞	∞	0,2	+1	+1,3

Tab. 2.5: Výstup SOVA pro druhý dekodér v první iteraci (BCH).

k	$L(u_k\|\underline{y})$	$L_c y_k$	$L(u_k)$	$L_e(u_k)$
1	+0,1	+0,9	+0,4	–1,2
2	–0,1	+0,8	+0,5	–1,4
3	–6,3	–3,8	+0,1	–2,6
4	+1,3	+0,2	+1,7	–1,1

Tab. 2.6: Výpočet vnější informace z druhého dekodéru v první iteraci (BCH).

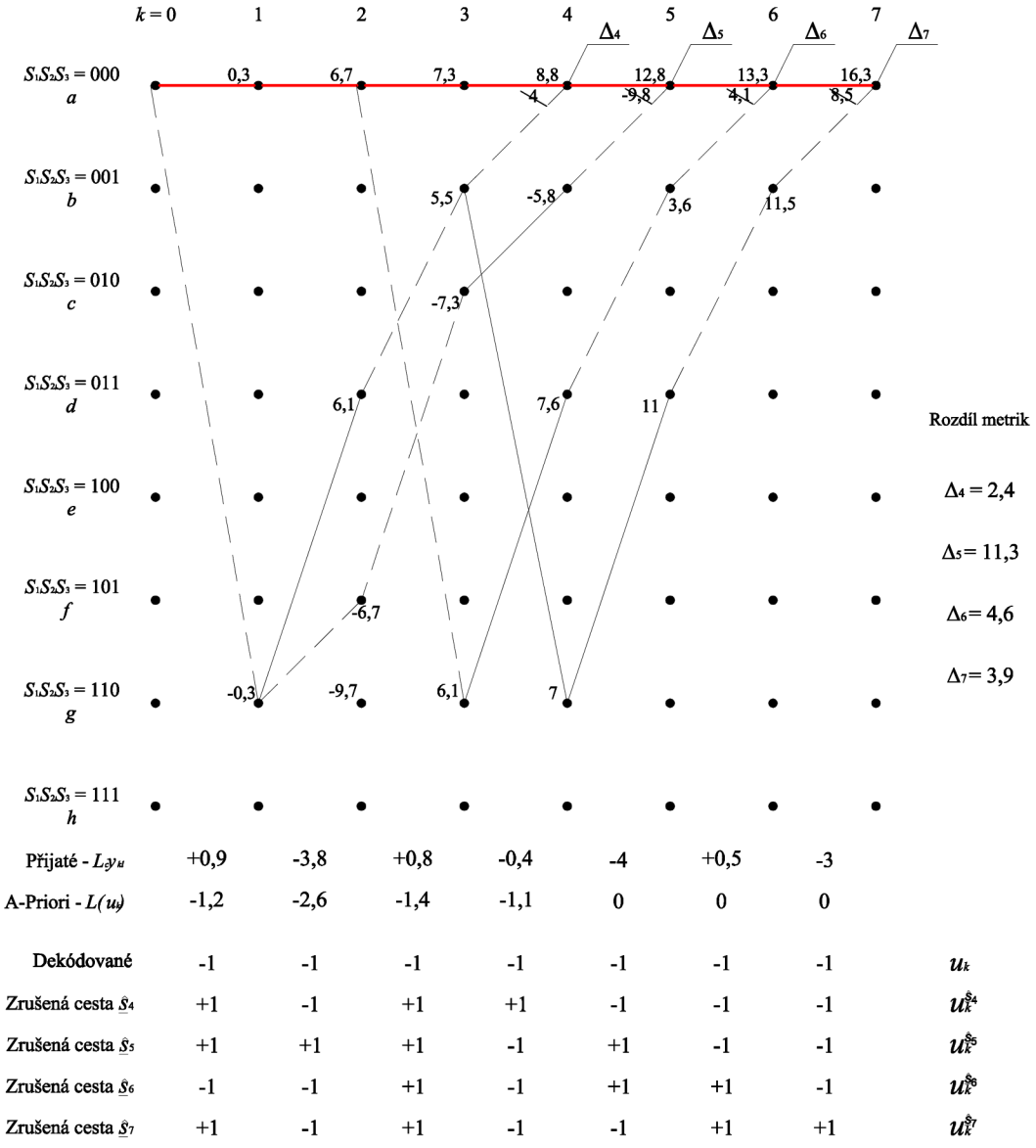
dekódovaná posloupnost je +1, −1, −1, +1 a −1, −1, +1, −1. Jediné shodné rozhodnutí o bitu je v $k = 2$, kde tedy máme $\left|L_{\widehat{\underline{s}_4}}(u_2)\right| = \infty$. Pro $k = 1$, 3 a 4 máme $\left|L_{\widehat{\underline{s}_4}}(u_1)\right| = \left|L_{\widehat{\underline{s}_4}}(u_3)\right| = \left|L_{\widehat{\underline{s}_4}}(u_4)\right| = 6,3$.

Podobné je to v $k = 5$ ve stavu b, kde přežívající a zrušená cesta mají dekódované bity +1, −1, −1, +1 a +1, +1, −1, −1, které jsou shodné pro $k = 1$ a 3. Z tohoto důvodu je v tab. 2.5 a-posteriori informace v $k = 1$ a 3 ∞, zatímco pro $k = 2$ a 4 se $\Delta_5 = (5,2 - 4,8)/2 = 0,2$. Zbývající hodnoty a-posteriori informace získáme stejným způsobem a jsou znázorněny v tab. 2.5. Nalezená minima nalezneme v 6. sloupci tab. 2.5. Podle vzorce 2.12 dostaneme a-posteriori hodnoty $L(u_k|\underline{y})$ dané vynásobením dekódovaných bitů u_k, $k = 1, \ldots, 4$ a výše uvedenými minimálními metrickými rozdíly. Tyto hodnoty jsou shrnuty v posledním sloupci tab. 2.5.

Třetí sloupec v tab. 2.6 znázorňuje přijaté vzorky $+0,9$, $+0,8$, $-3,8$, $-0,4$ a čtvrtý hodnoty vnitřní informace $+0,4$, $+0,5$, $+0,1$, $+1,7$ poskytnuté prvním dekodérem. V posledním sloupci jsou hodnoty vnější informace dané vzorcem 2.13.

Nyní srovnáme a-posteriori informaci $L(u_k|\underline{y})$ z druhého a prvního dekodéru. V prvním dekodéru, charakterizovaném v obr. 2.9, jsou tři chyby v datových bitech. Tyto tři chyby jsou redukovány druhým dekodérem (obr. 2.10) na dvě. Měkký výstup datových bitů u_1, u_2 a u_4 produkované druhým dekodérem, které jsou pro-

ložené datové bity u_1, u_3 a u_4 z prvního dekodéru, mají relativně nízkou hodnotu, která se blíží nule a tím signalizuje nízkou důvěru ve správnosti dekódování těchto bitů. Porovnáním hodnot v osmém sloupci tab. 2.3 a tab. 2.5 zjistíme, že hodnoty pravděpodobnosti správného dekódování byly druhým dekodérem zmenšeny.



Obr. 2.11: Mřížový diagram SOVA dekódování pro první dekodér v druhé iteraci (BCH).

V druhé iteraci použije první dekodér vnější informaci poskytnutou druhým dekodérem v předchozí iteraci. Na obr. 2.11 je znázorněn mřížový diagram pro první

k	$L_{\widehat{\underline{s}_k}}(u_k)$				Min	u_k	$L(u_k\|\underline{y})$
	$\left\|L_{\widehat{\underline{s}_4}}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}_5}}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}_6}}(u_k)\right\|$	$\left\|L_{\widehat{\underline{s}_7}}(u_k)\right\|$			
1	2,4	11,3	∞	3,9	2,4	–1	–2,4
2	∞	11,3	∞	∞	11,3	–1	–11,3
3	2,4	11,3	4,6	3,9	2,4	–1	–2,4
4	2,4	∞	∞	∞	2,4	–1	–2,4

Tab. 2.7: Výstup SOVA pro první dekodér v druhé iteraci (BCH).

dekodér v druhé iteraci. Můžeme zde vidět, že dekodér používá stejnou kanálovou posloupnost $L_c y_k$, jakou používal v první iteraci. Zvolená ML cesta je opět znázorněna tlustou červenou čarou. V tomto případě byla zvolena správně, a to jako cesta tvořená nulovými hodnotami. Znovu je vypočítána a-posteriori informace $L(u_k|\underline{y})$ a vnější informace $L_e(u_k)$, které jsou znázorněny v tab. 2.7 a tab. 2.8.

k	$L(u_k\|\underline{y})$	$L_c y_k$	$L(u_k)$	$L_e(u_k)$
1	–2,4	+0,9	–1,2	–2,1
2	–11,3	–3,8	–2,6	–4,9
3	–2,4	+0,8	–1.-,4	–1,8
4	–2,4	–0,4	–1,4	–0,9

Tab. 2.8: Výpočet vnější informace z prvního dekodéru v druhé iteraci (BCH).

Druhá iterace je pak dokončena nalezením vnější informace, generované prvním dekodérem, následným proložením a použita jako vnitřní informace pro druhý dekodér. Tento dekodér vybere opět nulovou cestu jako přežívající, a proto bude výstup turbo dekodéru po druhé iteraci správný.

3 IMPLEMENTACE V MATLABU

3.1 Úvod

V první části práce bylo teoreticky popsáno kódování a dekódování turbo konvolučních a turbo blokových kódů. Dalším úkolem bylo vytvořit demonstrační program, který by bylo možné využít jako výukovou pomůcku, a také jako analyzačního (simulačního) nástroje pro porovnávání dosažených parametrů kódu. Program byl vytvořen v programovém prostředí Matlab ve verzi R2010a [1][11][13][14][15].

Program obsahuje základní grafické rozhraní (*Turbo kódy*) pro volbu jednotlivých podprogramů, kde je možné zvolit dva výukové programy (*Výukový program TurboRSC, Výukový program TurboBCH*) a dva simulační programy (*TurboRSC grafy, TurboBCH grafy*), kde každá dvojice řeší problematiku turbo konvolučních nebo turbo blokových kódů.

Všechny programy využívají k dekódování SOVA [1] algoritmus a Log-MAP [5][11] algoritmus, který není předmětem této diplomové práce a byl přidán pouze za účelem srovnání. Dále je ve všech programech použit pseudonáhodný prokladač kap. 1.4.

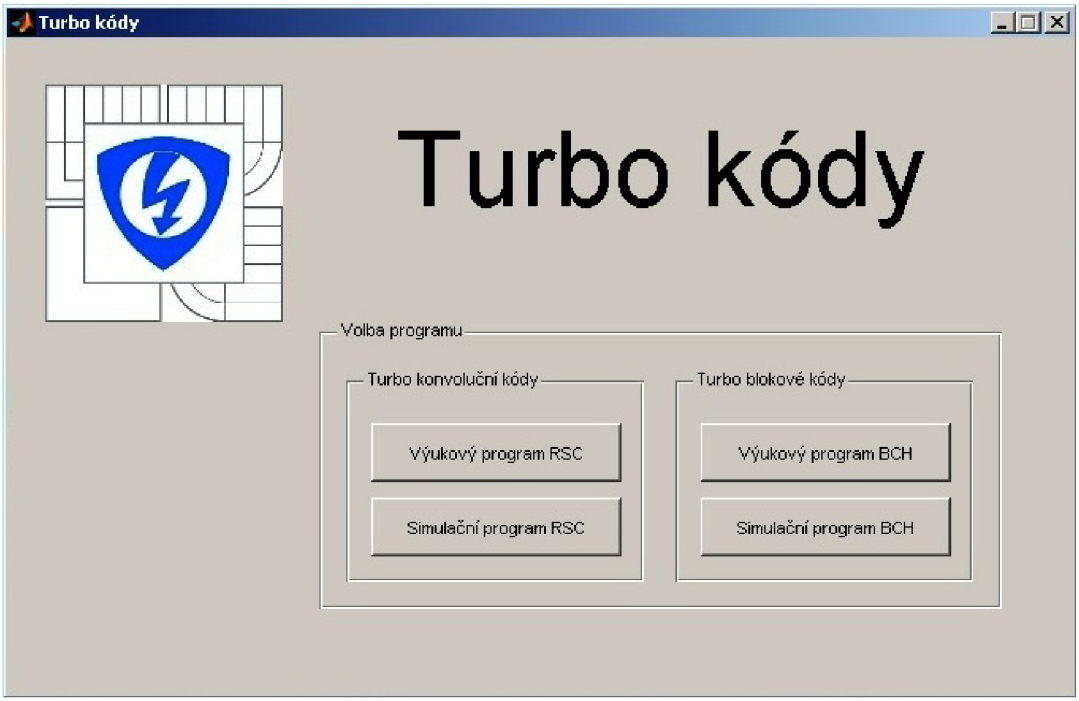
3.2 Popis programu

Vzhledem k velké podobnosti obou výukových a obou simulačních programů, je v této kapitole popsáno pouze jedno z každé dvojice grafických prostředí. Kapitola bude členěna do tří hlavních částí, kde bude popsáno hlavní grafické prostředí, jeden výukový a jeden simulační program.

3.2.1 Hlavní program

Na obr. 3.1 je zobrazen hlavní program, který byl vytvořen za účelem uvítacího menu. Je logicky členěn do dvou částí (*Turbo konvoluční kódy, Turbo blokové kódy*), ve kterých je možné zvolit jednotlivé podprogramy. Při volbě podprogramu (stisknutím příslušného tlačítka) je hlavní program automaticky uzavřen a následně zavolán

zpět až po uzavření zvoleného podprogramu.



Obr. 3.1: Hlavní program.

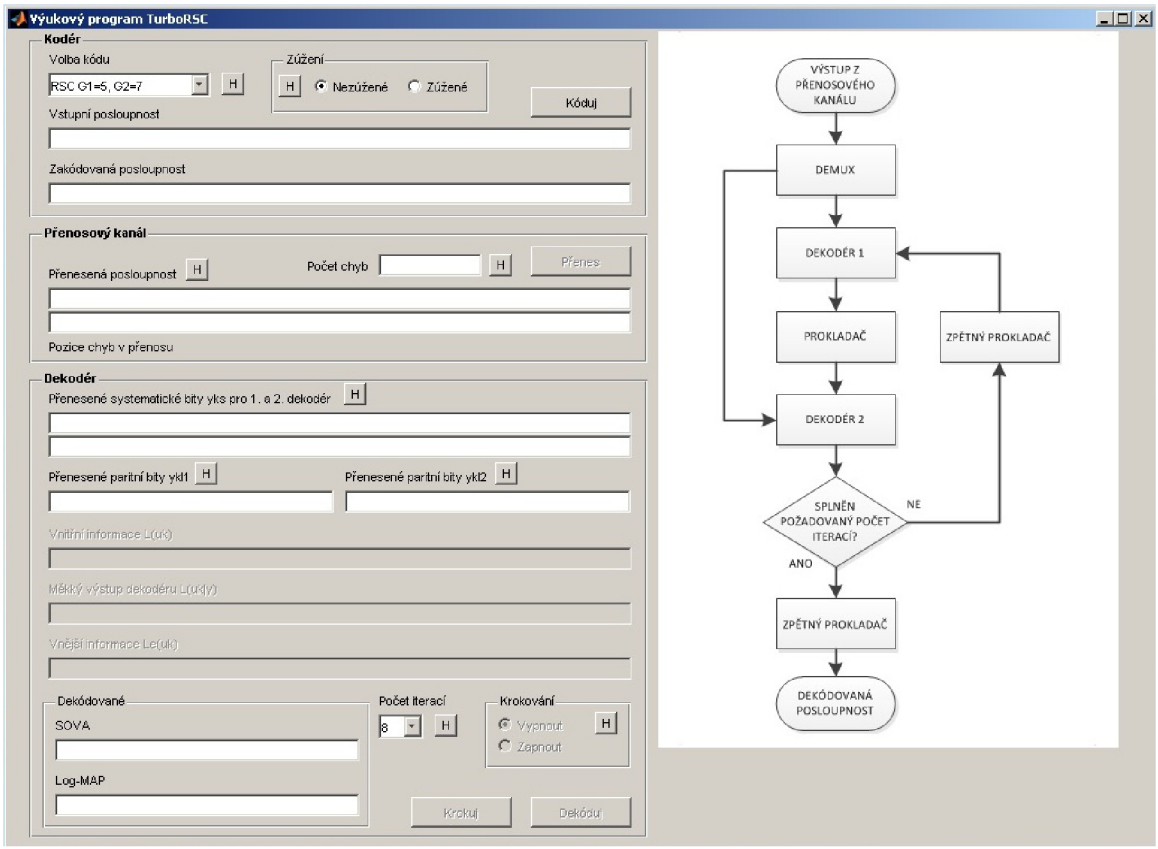
3.2.2 Výukový program

Samotný program obr. 3.2 se skládá ze tří hlavních bloků. Jedná se o bloky *Kodér, Přenosový kanál* a *Dekodér*. V následujícím textu bude vysvětlena funkce každého bloku a popsány možné vstupní parametry.

Blok Kodér

Tento blok obsahuje menu, pro který chceme kódování provést. V menu *Volba kódu* je možné zvolit pro turbo konvoluční kodér RSC kód daný vytvářecími mnohočleny $[G_0 = 7, G_1 = 5]$, $[G_0 = 15, G_1 = 17]$ nebo $[G_0 = 37, G_1 = 21]$. V případě volby výukového programu pro turbo blokové kódy je možné zvolit BCH (7, 4, 3), BCH (15, 11, 3) nebo BCH (31, 26, 2). Dále je zde nastavení použití zúžení (menu *Zúžení*). Tato volba je defaultně nastavena jako vypnutá tzn. všechny paritní bity z kodéru budou ponechány. Pokud volbu zúžení použijeme, je z každého kodéru odstraněn každý druhý paritní bit. Tato volba má výrazný vliv na informační rychlost

R. Posloupnost, kterou chceme zakódovat se vkládá do pole *Vstupní posloupnost*. Je nutné ji zadat v binární formě a jednotlivé bity oddělit mezerou. V případě ponechání prázdného pole program po stisknutí tlačítka *Kóduj* upozorní, že je pole prázdné a následně vygeneruje vstupní posloupnost pomocí funkce random. Následně je zpráva zakódována, multiplexována, vypsána do pole *Zakódovaná posloupnost* a předána do přenosového kanálu. Pořadí zobrazených zakódovaných bitů je popsáno v kap. 1.3 pro turbo konvoluční kódování a v kap. 2.5 pro turbo blokové kódy.



Obr. 3.2: Výukový program.

Blok Přenosový kanál

V tomto bloku je volitelná pouze jedna hodnota, a to zadání počtu chyb (pole *Počet chyb*), kterými bude přenášená posloupnost napadena. Není zde možné zadat hodnotu odstup signál/šum, tzn., že se nejedná o reálný přenosový kanál. Byl navržen

tak, aby co nejvíce vyhovoval výukovým účelům, což u reálného přenosového kanálu není možné docílit, protože nelze přesně definovat počet chyb. Při zmáčknutí tlačítka *Přenes*, je přenášená posloupnost napadena zadaným počtem chyb a následně zobrazena do pole *Přenesená posloupnost*. Pole *Pozice chyb v přenosu* může nabývat hodnot 0 a 1, kde 0 značí bit nenapadený chybou a 1 značí bit chybou napadený. Následně je posloupnost přenesených bitů předána bloku dekodéru.

Blok Dekodér

Hlavní pozornost při tvorbě výukového programu byla věnována právě tomuto bloku. Jsou zde dva hlavní volitelné parametry. První z nich je zadání počtu iterací, které dekodér provede. Tento parametr se zadává v menu *Počet iterací*. Jedná se o opakování dekódovacího procesu a tím i zpřesnění odhadu dekódovaných bitů. Druhý volitelný parametr je volba krokování (menu *Krokování*). V případě, že je krokování vypnuté, je aktivováno tlačítko *Dekóduj*. Po stisknutí tohoto tlačítka je provedeno demultiplexování přijaté posloupnosti z přenosového kanálu a vypsání demultiplexovaných bitů do příslušných polí (*Přenesené systematické bity y_{ks} pro 1. a 2. dekodér, Přenesené paritní bity y_{kl}^1* a *Přenesené paritní bity y_{kl}^2*). Dále se provede proces dekódování najednou pro požadovaný počet iterací. Výstupem je pak dekódovaná posloupnost pro oba dekódovací algoritmy, zobrazena v poli *Dekódované*.

Pokud je zvolena možnost krokování, aktivuje se tlačítko *Krokuj*. Touto volbou uživatel prochází jednotlivými kroky a v každém kroku dekódování jsou zobrazeny jednotlivé výsledky, vzorce výpočtů a pohyb diagramu dekódovacího procesu. Diagram je zobrazen v obr. 3.2. Krokování se provádí pouze pro dekodér používající SOVA algoritmus.
Zobrazované parametry během krokování:

- Demultiplexované systematické bity y_{ks} a paritní bity y_{kl} namapované pro přehlednost do logických 0 a 1.

- Vnitřní informace $L(u_k)$.

- Měkký výstup dekodéru $L(u_k|\underline{y})$.

- Vnější informace $L_e(u_k)$.

- Výstup dekodéru (dekódovaná posloupnost).

3.2.3 Simulační program

Tento program obr. 3.3 slouží jako analyzační nástroj pro porovnání dosažených parametrů kódu. Obsahuje pět bloků, ve kterých je možné nastavit pro kolik kódů chce uživatel simulaci provádět, parametry kodéru, parametry přenosového kanálu, parametry dekodéru a nastavení formátu uložení grafů. První čtyři jmenované nastavení mohou ovlivnit délku trvání simulace, která může dosáhnout až desítek hodin. To je také způsobeno tím, že Matlab je schopen k výpočtu simulací využít pouze jednoho jádra procesoru. Z tohoto důvodu je výhodnější použít na výpočet simulací procesor s vyšším taktem, než procesor s vyšším počtem jader.

Blok Nastavení simulace

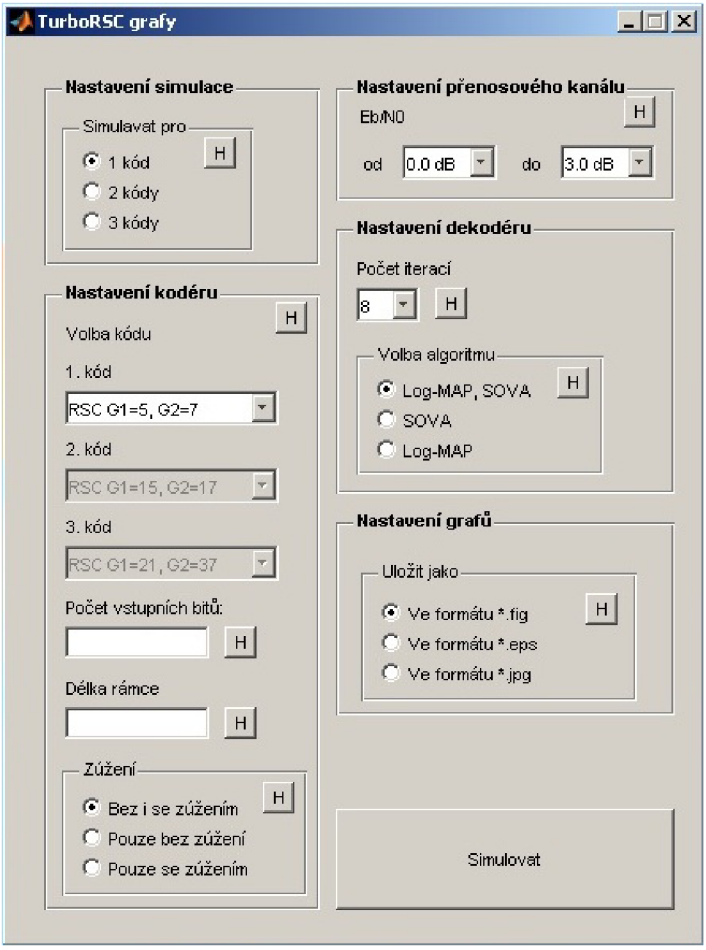
V tomto bloku je možné nastavit pro kolik kódů bude simulace provedena. Při měnění hodnot počtu kódů se automaticky aktivuje menu *Volba kódu*. V případě, že zvolíme simulaci pro 3 kódy, doba simulace nám naroste minimálně na trojnásobek.

Blok Nastavení kodéru

Zde je možné nastavit pro turbo konvoluční kódy až čtyři možné parametry. Pro turbo blokové kódy jsou parametry pouze tři, protože zde odpadá volba *Délka rámce*.

V menu *Volba kódu* je možné zvolit konkrétní kód, pro který bude simulace provedena. Volitelné kódy viz. 3.2.2. Při volbě velkých vytvářecích mnohočlenů délka simulace narůstá z důvodu zvětšujícího se mřížového diagramu při dekódování.

Pole *Počet vstupních bitů* slouží k zadání počtu bitů, které budou přivedeny do kodéru. Pro přesnější výstupní grafy je nutné zadat hodnotu minimálně 2000 bitů. Maximální hodnota je omezena na 100000 bitů. V případě ponechání prázdného pole



Obr. 3.3: Simulační program.

je uživatel při spuštění simulace na tuto skutečnost upozorněn a následně je hodnota nastavena na 1000 bitů. Hodnoty vstupních bitů jsou generovány automaticky programem pomocí funkce random.

Pole *Délka rámce* je zobrazena pouze pro turbo konvoluční kódy. Vstupní posloupnost je dělena do rámců o velikosti zadané hodnoty. Doporučená velikost rámce je 1000 - 5000 bitů. Pokud zůstane pole prázdné, program automaticky nastaví hodnotu na 1000 bitů.

Menu *Zúžení* viz. 3.2.2. V simulačním programu bylo menu *Zúžení* rozšířeno o možnost použít obě varianty. Při volbě této možnosti narůstá doba simulace na dvojnásobek.

Blok Nastavení přenosového kanálu

Zajišťuje nastavení přenosového kanálu, který odpovídá, na rozdíl od přenosového kanálu výukového programu, reálným podmínkám. Volitelný parametr je zde odstup signál/šum, kde si uživatel může zadat rozmezí, pro které bude simulace provedena. Defaultně jsou hodnoty nastaveny od 0dB do 3dB. Pokud jednu mezní hodnotu zvětšíme nebo zmenšíme o 1dB, délka simulace naroste nebo se zmenší o 1/4.

Blok Nastavení dekodéru

V tomto nastavení může uživatel měnit dva parametry. První z nich je *Počet iterací*, který udává kolikrát je blok nebo rámec přenesené zprávy dekódován. Čím vyšší je počet iterací, tím je odhad dekodéru přesnější. V praxi se používá rozmezí 4 - 14 iterací. Při použití vyššího počtu iterací se doba simulace zvyšuje.

Druhým parametrem je volba dekódovacího algoritmu (menu *Volba algoritmu*). Defaultně je volba nastavena pro oba algoritmy. Pokud zvolíme pouze SOVA algoritmus, doba simulace nám klesne téměř o 2/3, a to z důvodu, že Log-MAP algoritmus je mnohem náročnější na výpočet. Jak už bylo zmíněno v úvodu, tento algoritmus byl přidám pouze za účelem srovnání a není předmětem této diplomové práce.

Blok Nastavení grafů

Zde může uživatel zvolit v jakém formátu budou grafy uloženy. Doporučená a tedy defaultní volba je *.fig, z důvodu možné editace popisků grafů. Další možností je formát *.eps, kde se jedná o vektorovou grafiku. Poslední formát je zástupce bitmapového zobrazení *.jpg. Všechny grafy, nezávisle na zvoleném formátu, jsou po simulaci uloženy v případě zkompilované verze programu do složky src nebo nezkompilované verze přímo do rootu programu.

Výstup simulačního programu

Výstupem simulačního programu může být při největším rozsahu nastavení až 15 grafů, které vyjadřují závislost BER na E_b/N_0. V programu není přidána možnost nastavení, které grafy chce uživatel zobrazit. Tuto funkci si program obsluhuje sám.

Tedy pokud nastavíme simulaci pouze pro SOVA dekodér, grafy pro Log-MAP dekodér jsou automaticky programem zakázány a nezobrazí se ani neuloží.
Zobrazované grafy po simulaci:

- Graf 1-12 vynese závislost BER na E_b/N_0 pro všechny zvolené iterace SOVA a Log-MAP dekodéru pro zvolené kódy (zúžené i nezúžené).

- Graf 13 vynese porovnání závislosti BER na E_b/N_0 pro nejvyšší zvolenou iteraci SOVA a Log-MAP dekodéru pro zvolené kódy (zúžené i nezúžené).

- Graf 14 vynese porovnání závislosti BER na E_b/N_0 pro nejvyšší zvolenou iteraci SOVA a Log-MAP dekodéru pro zvolené kódy (pouze nezúžené).

- Graf 15 vynese porovnání závislosti BER na E_b/N_0 pro nejvyšší zvolenou iteraci SOVA a Log-MAP dekodéru pro zvolené kódy (pouze zúžené).

4 SIMULACE

4.1 Úvod

V této kapitole budou prezentovány výsledky simulací turbo kódů. Kapitola je členěna do dvou základních částí, kde v první části jsou zmíněny výsledky simulací pro turbo konvoluční kódy a v druhé výsledky turbo blokových kódů. Obě simulace používají AWGN přenosový kanál jako modulaci BPSK. Výkonnost turbo kódů je možné ovlivnit mnoha parametry. Jsou to parametry jako:

- Počet iterací, které provede dekodér při dekódování.
- Použití zúžení při kódování.
- Volba dekódovacího algoritmu.
- Délka rámce vstupních dat.
- Volba kódu.

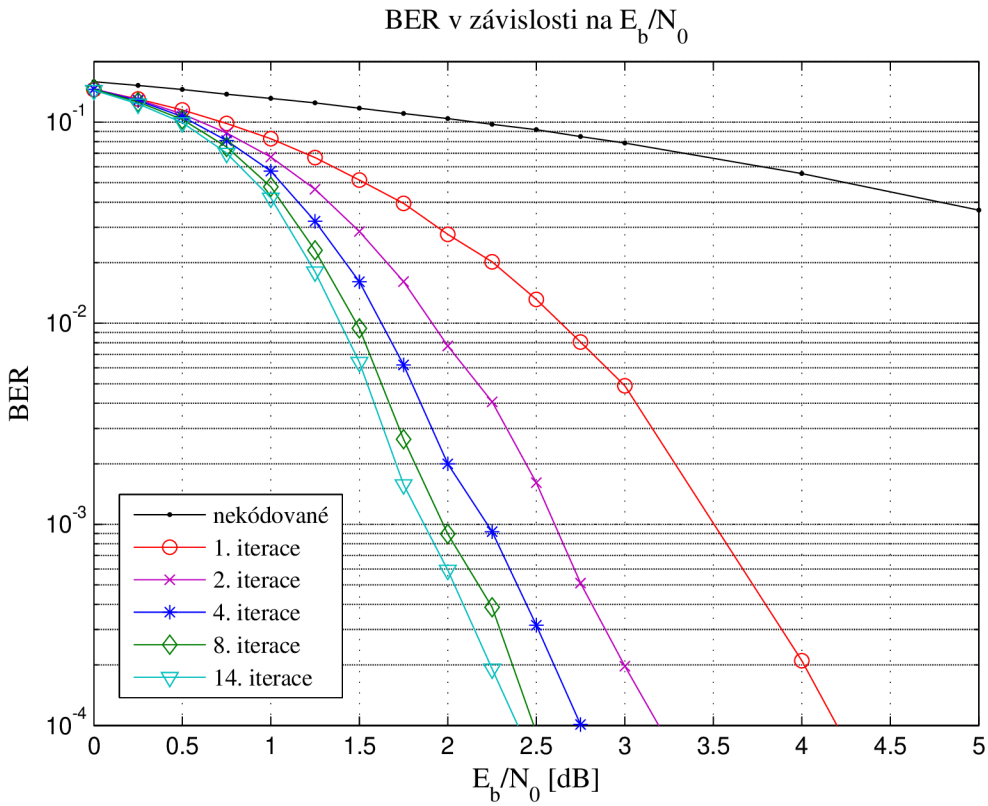
4.2 Turbo konvoluční kódy

V této části kapitoly budeme na základě simulací prezentovat vliv výše zmíněných parametrů na výkonnost turbo konvolučního kódu. Standardní parametry, které byly při simulaci použity, jsou znázorněny v tab. 4.1. Turbo kodér používá dva paralelně řazené kodéry. Jako kód byl zvolen RSC s vytvářecími mnohočleny $G_0 = 7$, $G_1 = 5$ a kódovým ohraničením $K = 3$. Jako prokladač byl zvolen pseudonáhodný s délkou $F = 1000$ bitů. Pokud není uvedeno jinak, bude vždy použito zúžení paritních bitů na polovinu a tím stoupne i informační rychlost $R = 1/2$. Defaultně je používán dekodér, který využívá k dekódování SOVA algoritmus. Zpravidla bylo použito 8 iterací při dekódování.

Kanál	AWGN
Modulace	BPSK
Kodér	Dva identické paralelně řazené RSC kodéry
Parametry RSC kódu	$n = 2$, $k = 1$, $K = 3$, $G_0 = 7$, $G_1 = 5$
Zúžení	Paritní bity z každého kodéru zúženy na polovinu $R = 1/2$
Dekodér	SOVA
P. iterací	8

Tab. 4.1: Parametry turbo konvolučního kodeku pro simulace.

4.2.1 Vliv počtu iterací

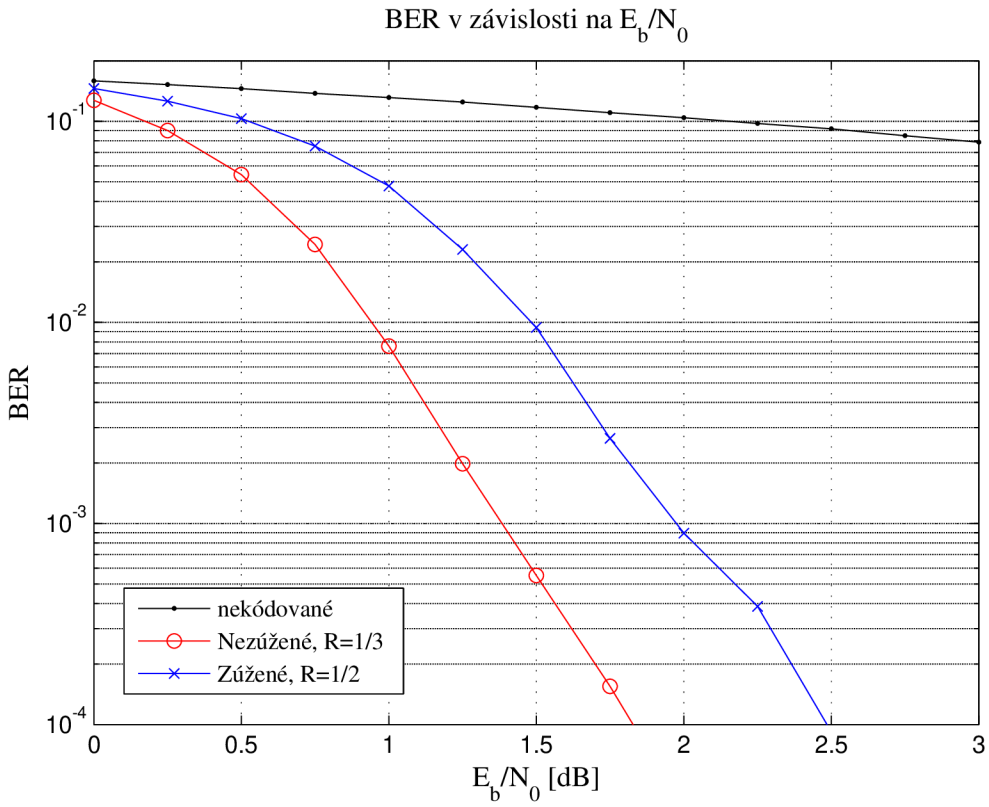


Obr. 4.1: Výkonnost turbo konvolučních kódů v závislosti na použitém počtu iterací dekodéru.

Obr. 4.1 zobrazuje výkonnost turbo konvolučního kódu v závislosti na použitém počtu iterací dekodéru. Pro srovnání je také v obrázku vynesena závislost pro nekódovanou posloupnost. Výkonnost po provedení první iterace dekódování by měla

být teoreticky srovnatelná s výkonností konvolučního kódu. S narůstajícím počtem iterací narůstá i výkonnost dekodéru. Například mezi první a druhou iterací je nárůst téměř o 1dB na hodnotě BER 10^{-4} a narůstá až do osmé iterace, kde je nárůst výkonnosti zpomalen. Mezi 8. a 14. iterací je kódový zisk na hodnotě BER 10^{-4} pouze o 0,1dB. Z obrázku je tedy možné vyvodit, že se zvyšujícím se počtem iterací narůstá výkonnost kódu, ale také výpočetní náročnost při dekódování. Proto je v praxi doporučená hodnota 4 - 14 iterací. Z tohoto důvodu je v následujících simulacích použito pouze 8 iterací dekodéru.

4.2.2 Vliv použití zúžení

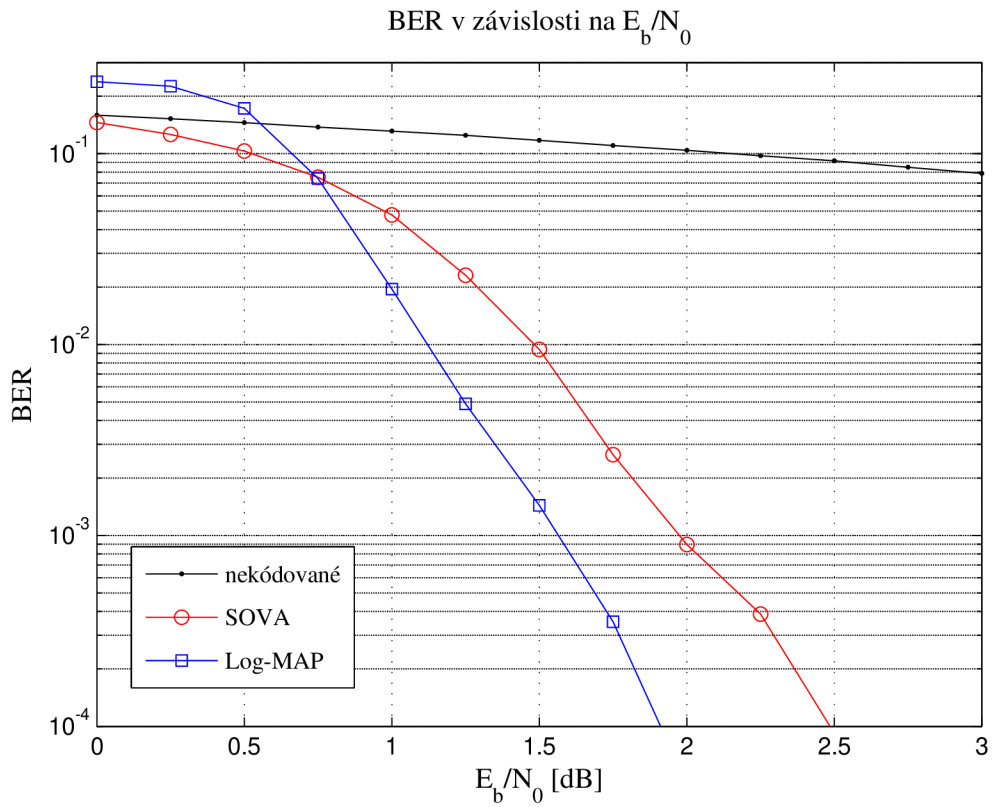


Obr. 4.2: Výkonnost turbo konvolučních kódů v závislosti na použití zúžení.

Jak už bylo zmíněno, v turbo kodéru se používá dvou nebo více kodérů, které při kódování produkují paritní bity. V simulaci jsou použity RSC kodéry. Jedná se o nejběžnější řešení, které je schopno dosáhnout informačního rychlosti pod 1/3. K tomu aby bylo možné dosáhnou informační rychlosti 1/2, je nutné každý druhý paritní bit

tzv. zúžit. Je také možné použít kód bez zúžení a tím ponechat informační rychlost na 1/3. Výkonnost takového kódu je zobrazena na obr. 4.2. Kódy používají stejné parametry jako v předchozí simulaci obr. 4.1. Turbo kodér pro nezúžený kód má na hodnotě BER 10^{-4} o 0,7dB větší zisk oproti turbo kodéru, který zúžení použil. Velmi podobných zisků je možné dosáhnout i pro jiné vytvářecí mnohočleny. Z obrázku je tedy zřejmé, že lepších výsledků dosáhneme v případě, že zúžení nepoužijeme. Jedná se však o zlepšení v řádech desetin decibelů. Vezmeme-li v úvahu menší počet přenášených bitů v případě zúžení, lze v některých případech považovat za výhodnější zúžení použít. Je také nutné si uvědomit, že zúžení nemá vliv na rychlost kódování a dekódování. Je pouze snížen počet přenášených bitů přenosovým kanálem a tím zmenšena přenosová šířka pásma.

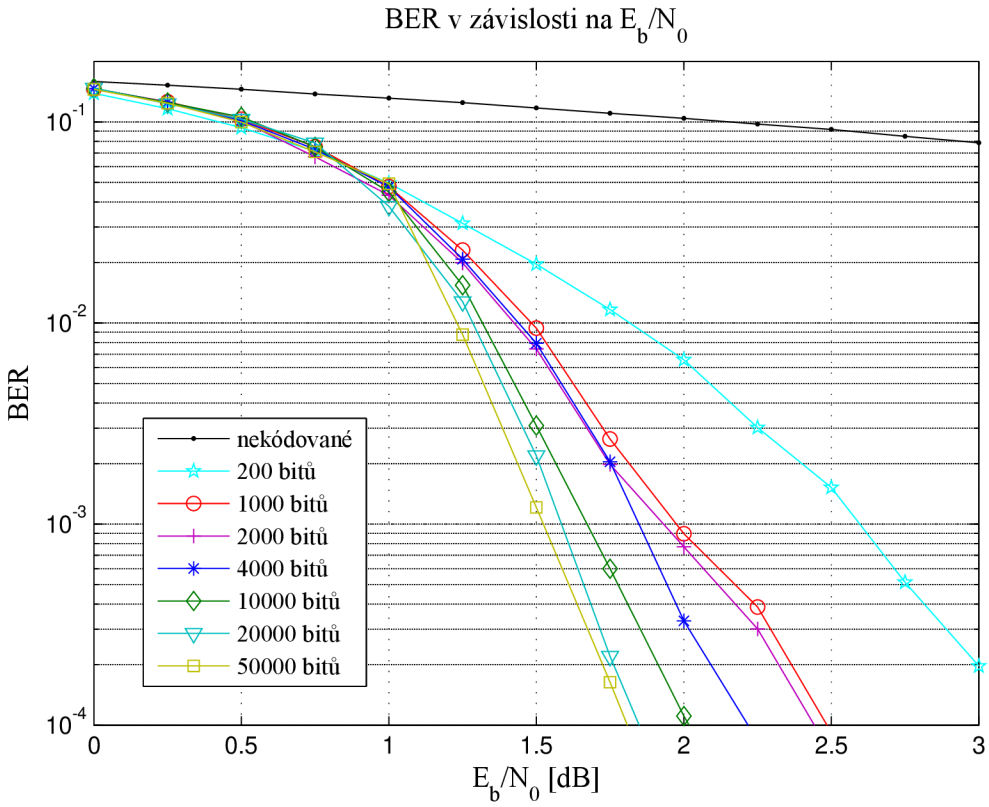
4.2.3 Vliv dekódovacího algoritmu



Obr. 4.3: Výkonnost turbo konvolučních kódů v závislosti na použitém dekódovacím algoritmu.

Na obr. 4.3 je zobrazeno porovnání výkonnosti turbo konvolučního kódu v závislosti na použitém dekódovacím algoritmu. Jedná se o algoritmus SOVA a algoritmus Log-MAP. Jak už bylo zmíněno, algoritmus Log-MAP byl do programu a simulací přidán pouze za účelem porovnání a není předmětem této diplomové práce. Pro oba dekodéry byly použity stejné parametry kódu jako v případě první simulace. Turbo kodér používající Log-MAP má kódový zisk vyšší na hodnotě BER 10^{-4} o 0,6dB. Dá se tedy konstatovat, že Log-MAP algoritmus je výkonnější oproti SOVA algoritmu. To samé už neplatí o jeho výpočetní náročnosti. Doba simulace byla pro Log-MAP téměř o 2/3 delší jak u SOVA algoritmu. To je také zapříčiněno tím, že SOVA prochází mřížový diagram pouze směrem dopředu, zatímco Log-MAP prochází mřížový diagram i zpětně. Z obrázku je tedy zřejmé, že Log-MAP dosahuje lepších výsledků, ovšem za cenu vyšší výpočetní náročnosti.

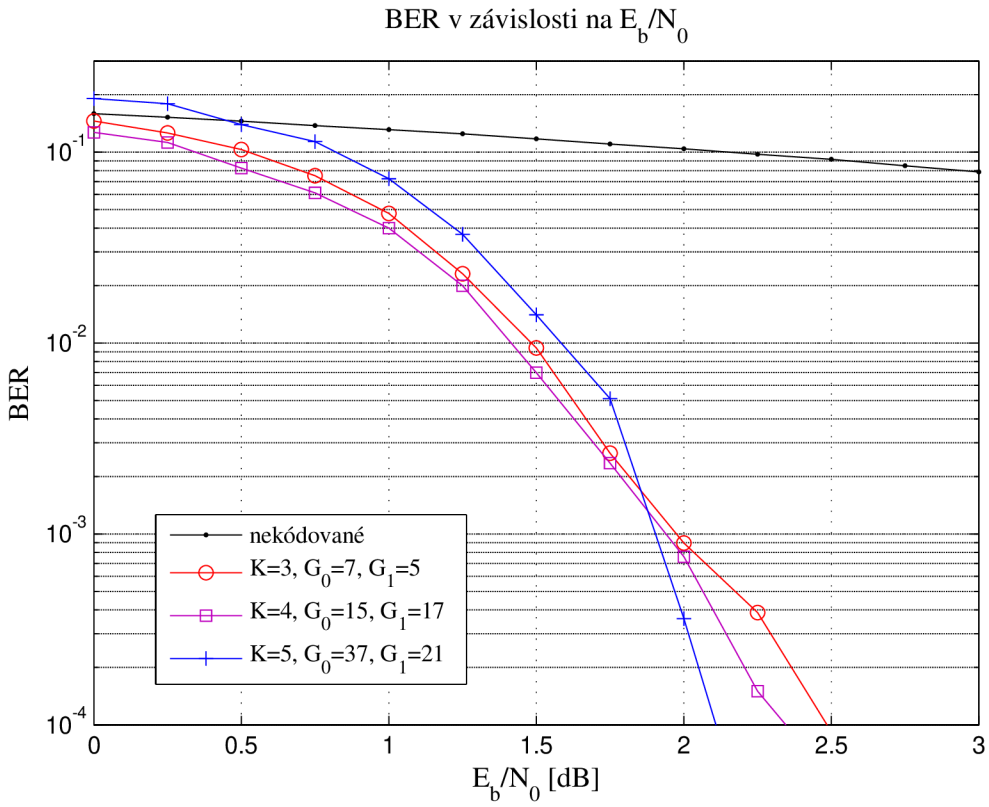
4.2.4 Vliv délky rámce



Obr. 4.4: Výkonnost turbo konvolučních kódů v závislosti na použité délce rámce.

Obr. 4.4 nám zobrazuje výkonnost turbo konvolučních kódů v závislosti na použité délce rámce. Pro mnoho aplikací, jako jsou například aplikace využívající real-time přenos, je velká délka rámce naprosto nepřijatelná. Rámce o délce 200 bitů jsou vhodné například pro přenos hlasu, 1000 bitů pak pro přenos videa. Systémy s větší délkou rámce jsou použitelné pro přenos dat a aplikace nevyžadující real-time přenos. Nejlepších výsledků v simulaci dosahoval turbo kód s délkou rámce 50000 bitů. Zlepšení výkonnosti je oproti rámci délky 1000 bitů na hodnotě BER 10^{-4} téměř 1dB. Z obrázku můžeme vyvodit, že s rostoucí délkou rámce výkonnost turbo konvolučních kódů stoupá, ale je také ovlivněno zpoždění, které dosahuje pro kratší rámce nižších hodnot.

4.2.5 Vliv volby kódu



Obr. 4.5: Výkonnost turbo konvolučních kódů v závislosti na zvoleném RSC kódu.

Na obr. 4.4 je zobrazena závislost výkonnosti turbo konvolučního kódu na zvoleném kódu. Jako první zástupce byl zvolen RSC kód, který byl použit pro všechny

simulace. Jedná se tedy o kód s vytvářecími mnohočleny $G_0 = 7$, $G_1 = 5$ a délkou kódového ohraničením $K = 3$. Druhým zvoleným kódem je $K = 4$, $G_0 = 15$, $G_1 = 17$. Turbo kód používající kód s kódovým ohraničením $K = 4$ dosahuje na hodnotě BER 10^{-4} o 0,1dB vyšší výkonnosti oproti kódu $K = 3$. Třetí zvolený kód má délku kódového ohraničení $K = 5$ a vytvářecí mnohočleny $G_0 = 37$, $G_1 = 21$. Pokud tento kód srovnáme s prvním kódem $K = 3$, dosahuje na hodnotě BER 10^{-4} o 0,4dB vyšší výkonnosti. Při srovnání s kódem $K = 4$ je nárůst výkonnosti o 0,25dB. Z obrázku je tedy zřejmé, že při volbě kódu s vyšším vytvářecím mnohočlenem můžeme dosáhnout vyšší výkonnosti turbo kódu. Jako protiklad této skutečnosti je nutné podotknout, že při větším vytvářecím mnohočlenu také narůstá velikost mřížového diagramu, což může ovlivnit výkon dekodéru, který pak pracuje s větším mřížovým diagramem.

4.3 Turbo blokové kódy

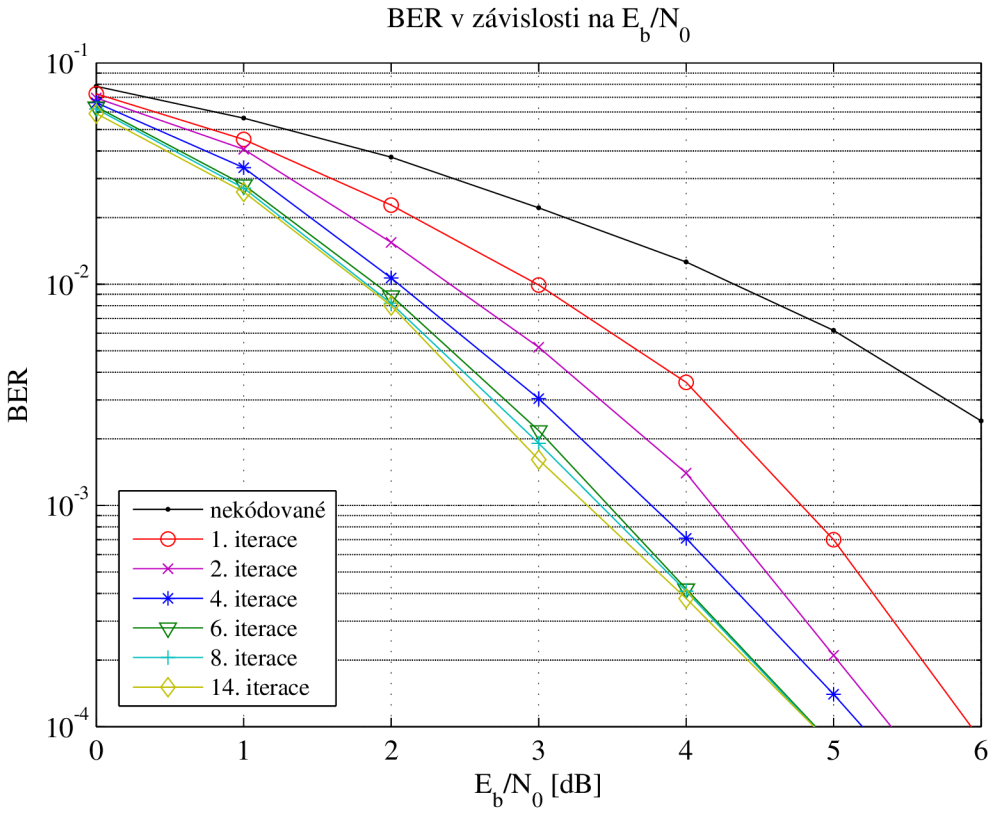
V této části se budeme věnovat simulacím provedených na základě výše zmíněných parametrů, které mohou ovlivnit výkonnost turbo blokových kódů. Základní nastavení kodeku pro tyto simulace je znázorněno v tab. 4.2. Jako kód byl zvolen nebinární BCH (31, 26, 3) kód. Uspořádání dílčích kodérů je paralelní a prokládání bylo zvoleno pseudonáhodné. Turbo kodér má informační rychlost $R = 0,72$ tzn., že nebylo aplikováno zúžení a všechny paritní bity budou přivedeny do přenosového kanálu. Dekodér používá k dekódování SOVA algoritmus a zpravidla bylo provedeno šest iterací dekódování.

Kanál	AWGN
Modulace	BPSK
Kodér	Dva identické paralelně řazené BCH kodéry
Parametry BCH kódu	$n = 31$, $k = 26$, $d_{min} = 3$,
Zúžení	není použito $R = 0,72$
Dekodér	SOVA
P. iterací	6

Tab. 4.2: Parametry turbo blokového kodeku pro simulace.

4.3.1 Vliv počtu iterací

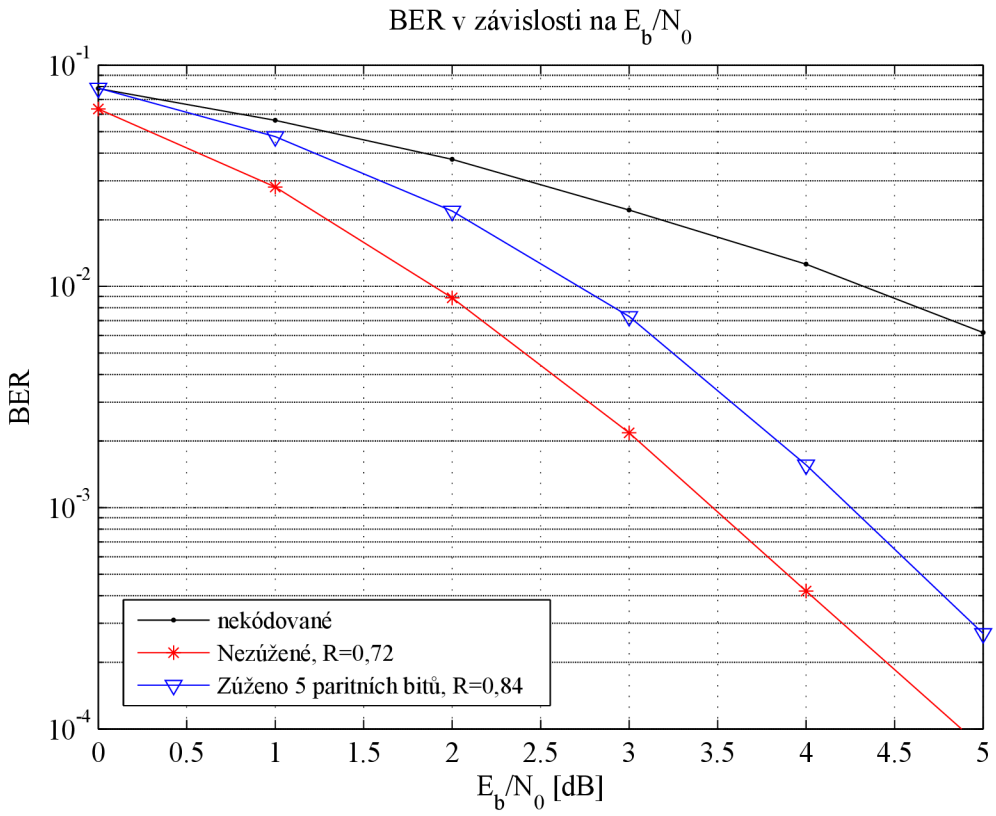
Na obr. 4.6 je znázorněna výkonnost turbo blokového kódu v závislosti na použitém počtu iterací dekodéru. Dále je v obrázku vynesena závislost pro nekódovanou posloupnost. S narůstajícím počtem iterací narůstá i výkonnost turbo blokového kódu, podobně jako tomu bylo v turbo konvolučních kódech. Rozdíl výkonnosti mezi první a druhou iterací na hodnotě BER 10^{-4} je 0,5dB. Jak narůstá počet iterací, rozdíl mezi jednotlivými iteracemi se snižuje až k šesté iteraci. Nárůst výkonnosti mezi 6. a 14. iterací je prakticky nulový. Z obr. 4.6 je tedy možné vyvodit, že nárůst výkonnosti od určitého počtu iterací je zanedbatelný stejně jako tomu bylo u turbo konvolučních kódu. V následujících simulacích budou v grafech vždy vynášeny šesté iterace dekódování.



Obr. 4.6: Výkonnost turbo blokových kódů v závislosti na použitém počtu iterací dekodéru.

4.3.2 Vliv použití zúžení

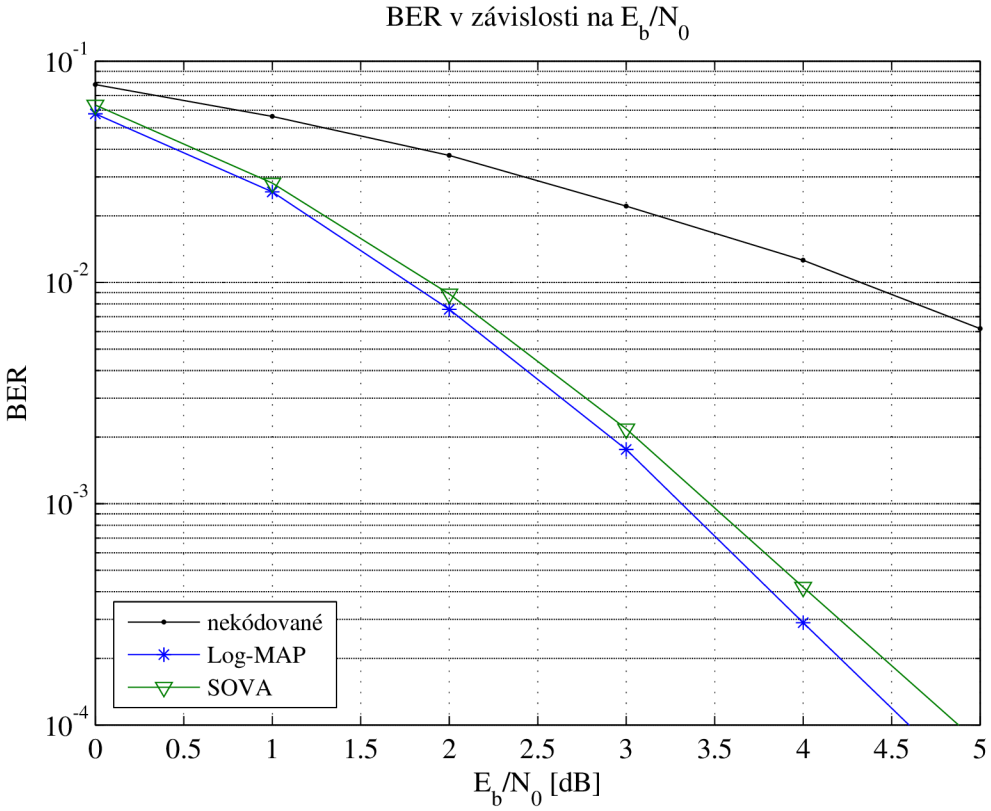
Jak už bylo zmíněno v simulaci pro turbo konvoluční kódy, je možné vlivem zúžení snížit počet paritních bitů před vstupem do přenosového kanálu. V našem případě byl počet paritních bitů snížen na polovinu, což ovlivnilo informační rychlost turbo blokového kodéru na hodnotu $R = 0,84$. Na obr. 4.7 je zobrazena výkonnost turbo blokového kódu v závislosti na použití zúžení. Rozdíl mezi turbo kódem používající zúžení a turbo kódem nepoužívající zúžení je na hodnotě BER 10^{-3} téměř 0,75dB. Může se zdát, že použití zúžení v případe turbo blokových kódů je výhodné jako u turbo konvolučních kódů. Pokud bychom porovnali výkonnost klasického BCH (31, 26, 3) [5] kódu pak, by výkonnost turbo blokového kódu za použití zúžení byla minimálně srovnatelná nebo horší jak u klasického BCH kódu.



Obr. 4.7: Výkonnost turbo blokových kódů v závislosti na použití zúžení.

4.3.3 Vliv dekódovacího algoritmu

Na obr. 4.8 je zobrazeno porovnání výkonnosti turbo blokového kódu v závislosti na použitém dekódovacím algoritmu. Jako v případě turbo konvolučních kódů se jedná o algoritmus SOVA a algoritmus Log-MAP. Log-MAP dosahuje lepší výkonnosti jak SOVA, a to o 0,3dB na hodnotě BER 10^{-4}. Simulace se potýkala se stejným problémem, jako simulace turbo konvolučních kódů. Výpočetní náročnost je pro Log-MAP podstatně větší jak pro SOVA a nárůst doby simulace byl téměř o 2/3.

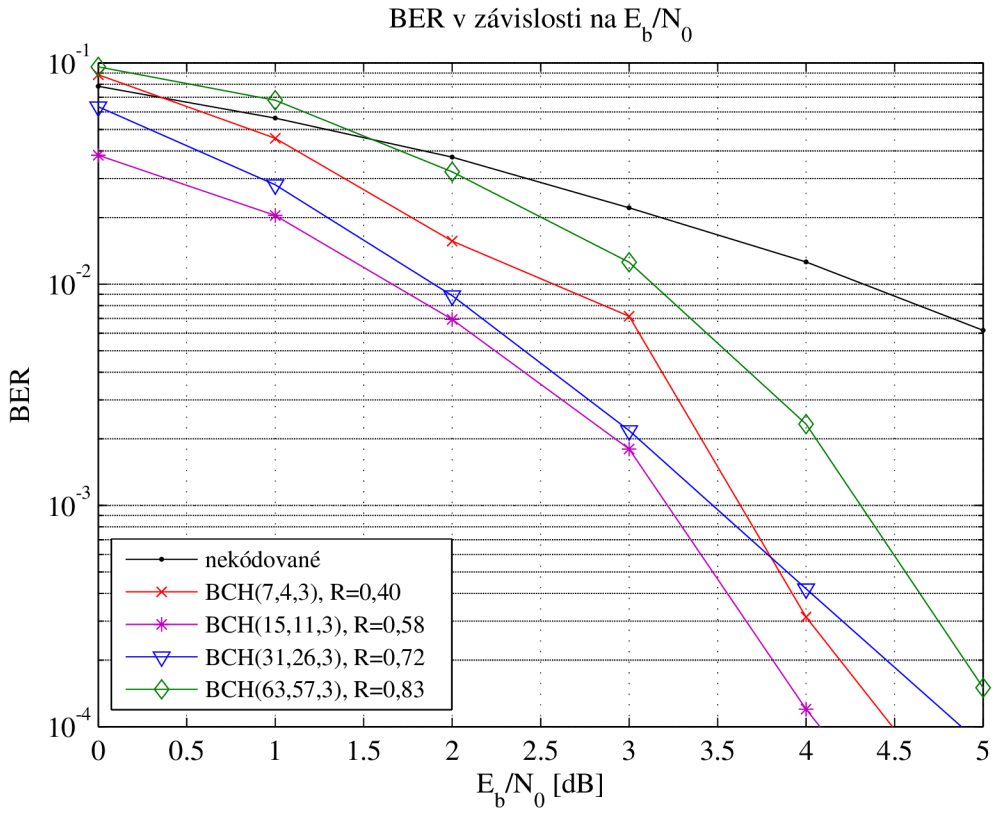


Obr. 4.8: Výkonnost turbo blokových kódů v závislosti na použitém dekódovacím algoritmu.

4.3.4 Vliv volby kódu

Obr. 4.9 zobrazuje závislost turbo blokového kódu na zvoleném BCH kódu. Jako první kód byl zvolen BCH (7, 4, 3) s informační rychlostí $R = 0,4$. Tento kód dosáhl hodnoty BER 10^{-4} na hodnotě E_b/N_0 4,5dB. Dalším zvoleným kódem je

BCH (63, 57, 3) s informační rychlostí $R = 0,83$ a pro tento kód dosahoval turbo kodér nejhorších výsledků. Na hodnotu BER 10^{-4} dosáhl až na E_b/N_0 5,2dB. Třetí zvolený kód by byl použit pro všechny simulace a jedná se o kód BCH (31, 26, 3) s informační rychlostí $R = 0,72$, který dosahuje hodnot BER 10^{-4} na hodnotě E_b/N_0 4,8dB. Poslední zvoleným kódem je BCH(15,11,3) s informační rychlostí $R = 0,58$. Turbo kód používající tento kód dosahoval v simulaci nejlepší výkonnosti, která byla na hodnotě BER 10^{-4} o 0,4dB od BCH (7, 4, 3), 0,8dB od BCH (31, 26, 3) a 1,1dB od BCH (63, 57, 3).



Obr. 4.9: Výkonnost turbo blokových kódů v závislosti na zvoleném BCH kódu.

4.4 Zhodnocení výsledků simulací

V této kapitole byly prezentovány výsledky simulací, provedené na vytvořeném programu v programovém prostředí Matlab. Srovnáme-li výsledky s jinými, například v literatuře [5], můžeme říci, že téměř odpovídají teoretickým předpokladům.

S rostoucím počtem provedených iterací dekodéru stoupá výkonnost turbo kódů. U turbo konvolučních kódů je od osmé iterace tento rozdíl zanedbatelný. V případě turbo blokových kódů dochází ke snížení růstu výkonnosti o dvě iterace dříve, tedy v šesté. Proto je v praxi doporučená hodnota iterací 4 - 14.

Při použití zúžení dochází ke snížení výkonnosti turbo kódů. Rozdíl však není velký a výhodou je zmenšení přenosové šířky pásma. U turbo konvolučních kódů se dá říci, že je tento pokles téměř zanedbatelný. Turbo blokové kódy se potýkají s problémem, že v některých případech je výkonnost při použití zúžení minimálně srovnatelná s výkonností běžného BCH kódu.

Volba kódu má rovněž vliv na výkonnost v případě, že zvolíme kód s větším vytvářecím mnohočlenem, dosáhneme tím lepší zabezpečovací schopnosti. Z toho plyne nevýhoda rostoucího mřížového diagramu, a ta se projeví při dekódování, které pak musí procházet větší diagram a tím se zvyšuje výpočetní náročnost.

Dalším parametrem ovlivňujícím výkonnost je délka rámce. Při použití velké délky rámce dosáhneme zvýšení výkonnosti, ale také narůstání zpoždění, které by bylo například pro real-time aplikace kritické.

Poslední parametrem je pak volba dekódovacího algoritmu. Turbo kódy dosahují lepší výkonnosti za použití Log-MAP algoritmu než, za použití dekódovaní pomocí SOVA. Nevýhodou je pak vyšší výpočetní náročnost dekodéru. V některých případech byla simulace delší o 2/3.

5 ZÁVĚR

První fází diplomové práce bylo nastudovat problematiku protichybového zabezpečení pomocí turbo konvolučních a turbo blokových kódu. Dále se zaměřit na problematiku dekódování zprávy zabezpečené uvedenými kódy.

Problematice turbo konvolučních kódů je věnována první kapitola. Je zde popsáno kódování pomocí RSC kódů a jejich aplikace do turbo kodéru. Jsou zde uvedeny základní pojmy jako je prokládání a zúžení. Dále jsou uvedeny matematické vztahy pro dekódovací algoritmus SOVA, popsáno iterativní dekódování, které je doplněno podrobný příkladem. Problematiku turbo blokových kódů řeší kapitola druhá, kde je vysvětlen postup kódování pomocí binárních BCH kódů, a také jejich aplikace do turbo blokového kodéru. Stejně jako v první kapitole jsou zde uvedeny matematické vztahy pro SOVA algoritmus a jeho modifikace, rozebrán princip iterativního dekódování turbo blokových kódů a vše je názorně vysvětleno na uvedeném příkladu.

Druhá fáze diplomové práce je zaměřena na vytvoření demonstračního programu v programovém prostředí Matlab, který bude možno využít jako výukovou pomůcku a rovněž jako analyzačního nástroje pro porovnání dosažených parametrů kódu. S pomocí vytvořeného programu realizovat zevrubné srovnání dosažených parametrů pro různé parametry kódu.

Demonstračnímu programu je věnována třetí kapitola. Program je členěn na čtyři grafické prostředí, kde dvě jsou navrženy tak, aby bylo možné je využít jako výukovou pomůcku a jednotlivé položky jsou doplněny nápovědou. Následující dvě grafické prostředí jsou koncipovány jako simulační programy, pomocí kterých je možné analyzovat výkonnost turbo kódů. Kodek je navržen tak, aby se co nejvíce blížil reálným podmínkám. V poslední části, a tedy čtvrté kapitole jsou zobrazeny výsledky simulací, a také porovnání vlivu jednotlivých parametrů na výkonnost turbo kódů. Celkově lze říci, že bylo dosaženo výsledků odpovídajících různým zdrojům. Závěry z těchto simulací lze shrnout následovně: výkonnost turbo kódu roste se zvyšujícím se počtem provedených iterací dekódování, vhodnou volbou kódu (vytvářecího mnohočlenu), změnou délky rámce, nebo také použitím jiného dekódovacího algo-

ritmu. Výkonnost turbo kódů naopak klesá při použití zúžení. Je tedy možné sestavit kodek s vysokou výkonností. Daní je však rostoucí náročnost, ať už se jedná o časovou (vliv počtu iterací) nebo výpočetní (ovlivněna volbou kódu nebo dekódovacího algoritmu). Věřím, že tímto byly splněny všechny cíle zadání.

LITERATURA

[1] BOHDANOWIC, A. Soft Input Soft Output Viterbi Algorithm/ [online]. 2003. poslední aktualizace 1. 8. 2003 [cit. 13. 5. 2011]. Dostupné z URL: http://www.mathworks.com/matlabcentral/fileexchange/3801-soft-input-soft-output-viterbi-algorithm

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

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

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

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

[6] HUANG, F. Turbo Code Encoder/ [online]. 1997. poslední aktualizace 29. 5. 1997 [cit. 12. 2. 2011]. Dostupné z URL: http://http://scholar.lib.vt.edu/theses/available/etd-71897-15815/unrestricted/chap3.pdf

[7] HUANG, F. Iterative Turbo Decoder/ [online]. 1997. poslední aktualizace 29. 5. 1997 [cit. 12. 2. 2011]. Dostupné z URL: http://scholar.lib.vt.edu/theses/available/etd-71897-15815/unrestricted/chap4.pdf

[8] LEE, L.H.C. Convolutional Coding - Fundamentals and Applications. Artec House, 1997, ISBN 0-89006-914-X.

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

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

[11] MORTHY, H. Log MAP Decoder/ [online]. 2009. poslední aktualizace 24. 7. 2009 [cit. 13. 5. 2011]. Dostupné z URL: http://www.mathworks.com/matlabcentral/fileexchange/24848-log-map-decoder

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

[13] ZAPLATÍLEK, K., DOŇAR, B. MATLAB - pro začátečníky BEN - technická literatura, Praha 2004, ISBN 80-7300-175-6

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

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

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

AWGN aditivní bílý Gaussův šum – Aditive White Gaussian Noise

BCH Bose-Chaudhuri-Hocquenghem

BER bitová chybovost – Bit Error Rate

BPSK binární fázové klíčování – Binary Phase Shift Keying

MAP Maximum A-Posteriori

ML maximálně pravděpodobná – Maximum Likelihood

MSB nejvíce významný bit – Most Significant Bit

NRC nerekurzivní konvoluční – Non-Recursive Convolutional

LLR logaritmicko pravděpodobnostní poměr – Log Likelihood Ratio

LSB nejméně významný bit – Least Significant Bit

RSC rekurzivní systematický konvoluční – Recursive Systematic Convolutional

SOVA měkký výstup Viterbiho algoritmu – Soft-Output Viterbi Algorithm

XOR eXclusive OR

$c(x)$ kódové slovo

Δ_k rozdíl metrik

δ přechod v mřížovém diagramu

$d(x)$ datové bity

d_{min} minimální Hammingova vzdálenost

E_b střední energie na jeden bit přenášené informace

$\underline{e}_k$ update sekvence

$\mathbf{G}$ vytvářecí mnohočlen

$\mathbf{J}$ zúžovací matice

K délka kódového ohraničení

k časový index (pozice v mřížovém diagramu)

$L_e(u_k)$ vnější informace

L_c spolehlivost přenosového kanálu

$L_c y$ měkký výstup přenosového kanálu

$L_c y_{kl}$ paritní bity z přenosového kanálu

$L_c y_{ks}$ systematické (datové) bity z přenosového kanálu

$L(u_k)$ a-priori informace (vnitřní informace)

$L(u_k \mid \underline{y})$ a-posteriori informace (měkký výstup)

m počet paměťových buněk kodéru

$M(s_k)$ metrika cesty

n výstupní bity

N_0 spektrální výkonová hustota šumu

Π Matice pseudonáhodného prokladače

P pravděpodobnost přechodu

R informační rychlost

S_k stav paměťové buňky kodéru, stav v mřížovém diagramu

s_k cesta v mřížovém diagramu

u_k vstupní, výstupní (dekódovaný) bit

x_k přenášený bit

x_{kl} přenášená posloupnost (výstup z turbo kodéru)

$\underline{y}$ přijatá posloupnost

y_{kl} paritní bity

y_{ks} systematické (datové) bity

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

Na přiloženém CD jsou uloženy zdrojové kódy pro výukový a simulační program vytvořené v prostředí Matlab (M-file). Tyto soubory se nacházejí ve složce //Matlab /Turbokody_v2.3a_zdrojove_kody/. Dále je zde také zkompilovaná verze programu do *.exe (//Matlab/Turbo_kody_v2.3b/src/). Aby byl tento program spustitelný je nutné mít v PC nainstalovaný Matlab nebo nainstalovat přiložený balíček MCRInstaller (//Matlab/MCRInstaller.exe), který zajistí spustitelnost programu v kompilované verzi bez nutnosti instalace Matlabu. Dále je přiložen text diplomové práce ve formátu *.pdf a zdrojové kódy textu psané v LaTeXu (//Latex/xsedyj00/). Poslední složkou na přiloženém CD je složka obrázky, která obsahuje všechny obrázky použité v diplomové práci ve formátu *.dwg (AutoCAD).