VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020

Bc. Radek Závorka

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY
DEPARTMENT OF RADIOENGINEERING

PROGRAM PRO DEMONSTRACI KANÁLOVÉHO KÓDOVÁNÍ
PROGRAMME FOR CHANNEL CODING DEMONSTRATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Radek Závorka**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. Aleš Prokeš, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Elektronika a sdělovací technika**

Ústav radioelektroniky

Student: Bc. Radek Závorka ***ID:*** 186242

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Program pro demonstraci kanálového kódování

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vlastnostmi základních blokových, konvolučních a zřetězených kanálových kódů. Prostudujte používané metody kódování a dekódování. Ve vhodném programovacím prostředí (preferován je Matlab) vytvořte sadu funkcí pro názornou demonstraci činnosti kanálových kodérů a dekodérů s jednoduchými grafickými výstupy. Navrhněte a vytvořte model přenosového AWGN kanálu s mnohacestným šířením.

Vytvořte vhodné uživatelské prostředí pro snadné ovládání demonstračního programu (například GUI při použití Matlabu). Je požadována možnost nastavení typu kódu, struktury vstupních dat, nastavení rozložení chyb v přijaté posloupnosti nebo začlenění určitého typu kanálu spolu s nastavením jeho parametrů. Požadovanými činnostmi programu jsou pak určení pozice chyb, jejich oprava a vyhodnocení chybovosti v závislosti na parametrech kanálu. Vytvořte návod pro počítačové cvičení.

DOPORUČENÁ LITERATURA:

[1] PROKEŠ, A. Rádiové komunikační systémy. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. 2013.

[2] ODENWALDER, J. P. Error Control Coding Handbook. San Diego: Linkabit Corporation, 1976.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 28.5.2020

Vedoucí práce: prof. Ing. Aleš Prokeš, Ph.D.

prof. Ing. Tomáš Kratochvíl, Ph.D.
předseda oborové rady

ABSTRAKT

Cílem práce je vytvořit program pro demonstraci kanálového kódování využitelný při výuce. Byly vybrány kódy od jednodušších až po složitější, které se blíží limitu Shannonova teorému o kapacitě kanálu. Jedná se o Hammingův kód, cyklický kód, konvoluční kód a LDPC kód. Potřebné funkce vycházející z teoretického základu, který je v práci podrobně rozebrán, byly napsány v programovacím jazyce Matlab. Jako výstup slouží uživatelské rozhraní, kde je možno zadávat informační slovo, simulovat průchod přenosovým kanálem a názorně sledovat, jak kódování a dekódování u jednotlivých kódů probíhá. Obsahem práce je také srovnání jednotlivých kódů z hlediska bitové chybovosti v závislosti na poměru SNR a možných parametrech. Závěrem je přiložen návod pro počítačové cvičení, jehož obsahem je nutná teorie, zadání a připravené tabulky pro snadné vypracování požadovaných úkolů.

KLÍČOVÁ SLOVA

kanálové kódy, Hammingovy kódy, cyklické kódy, konvoluční kódy, LDPC kódy, kodér, dekodér, program, Matlab

ABSTRACT

The main subject of this thesis is creating a programme, used for channel coding demonstration. This programme will be used for teaching purposes. The programme contains various codes from simple ones, to those which almost reach Shanon's channel capacity theorem. Specifically these are the Hamming code, cyclic code, convolutional code and LDPC code. These functions are based on theoretical background described in this thesis and have been programed in Matlab. Practical output of this thesis is user interface, where the user is able to input information word, simulate transmission through the transmission channel and observe coding and decoding for each code. This thesis also contains a comparison between individual codes, concerning bit-error rate depending on SNR and various parameters. There is a computer lab with theoretical background, assignment and sheets for convenient accomplishment of each task.

KEYWORDS

channel codes, Hamming code, cyclic code, convolutional code, LDPC code, encoder, decoder, program, Matlab

ZÁVORKA, Radek. *Program pro demonstraci kanálového kódování*. Brno, 2020, 85 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky. Vedoucí práce: prof. Ing. Aleš Prokeš, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Program pro demonstraci kanálového kódování“ 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/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

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

Obsah

Seznam obrázků

Seznam tabulek

Úvod

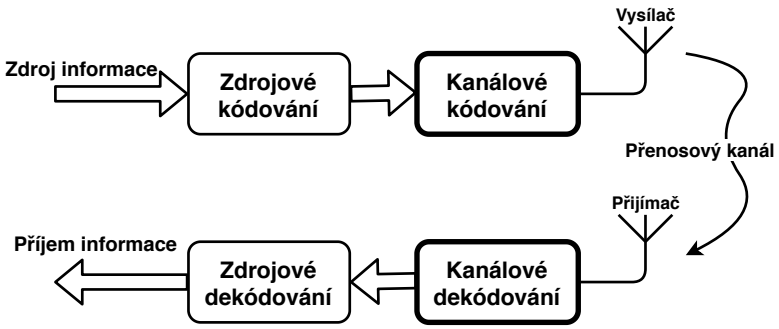
Cílem diplomové práce je vytvořit program pro demonstraci kanálového kódování. V reálném přenosovém řetězci působí na přenášená data mnoho nežádoucích vlivů, které způsobují chyby v přenosu. Jedná se o přídavný šum, mnohacestné šíření, přeslechy a další. Cílem kanálového kódování je zabezpečit data proti chybám při přenosu. Existuje mnoho metod a kódů, jež umí chybu po průchodu přenosovým kanálem nejen detekovat, ale dokonce i opravit. Podle toho se kódy nazývají detekční, nebo korekční.

Program má sloužit jako učební pomůcka, proto byly pro simulaci funkce vybrány kódy od základních až po složitější, které se blíží limitu Shannon-Harleyova vztahu o kapacitě kanálu. Konkrétně se jedná o Hammingův kód, cyklický kód, konvoluční (Viterbiho) kód a kód s řídkou paritní maticí - Low Density Parity Check (LDPC). V první kapitole práce jsou podrobně popsány metody kódování a dekódování pro všechny výše zmíněné kódy, včetně názorných příkladů. V programovacím prostředí Matlab byly napsány funkce pro názornou demonstraci činnosti a vytvořeno uživatelské rozhraní pro snadné ovládání, přehlednější prezentaci výsledků a možnost pochopení principů jednotlivých kódů. Ovládání programu je popsáno ve druhé kapitole, kde je pomocí náhledu obrazovek podán přehledný výklad dosažených výsledků.

Další, tj. třetí kapitola je věnována porovnání diskutovaných kódů především s ohledem na jejich bitovou chybovost BER v závislosti na poměru signálu k šumu (SNR). Poslední kapitola je zaměřena na vytvoření návodu pro počítačové cvičení, obsahující nezbytnou teorii a zadání, včetně tabulek pro přehledné vypracování.

1 Teoretická část

V této kapitole budou popsány jednotlivé kanálové kódy, jejich funkce, princip kódování a dekódování a vše bude názorně předvedeno na příkladech. Než však přistoupíme k samotným kódům, je vhodné přiložit obrázek 1.1, který zobrazuje blokové schéma komunikačního kanálu, na němž je jasně vidět, kde se kanálový kodér a dekodér nachází.



Obr. 1.1: Blokové schéma zjednodušeného komunikačního kanálu.

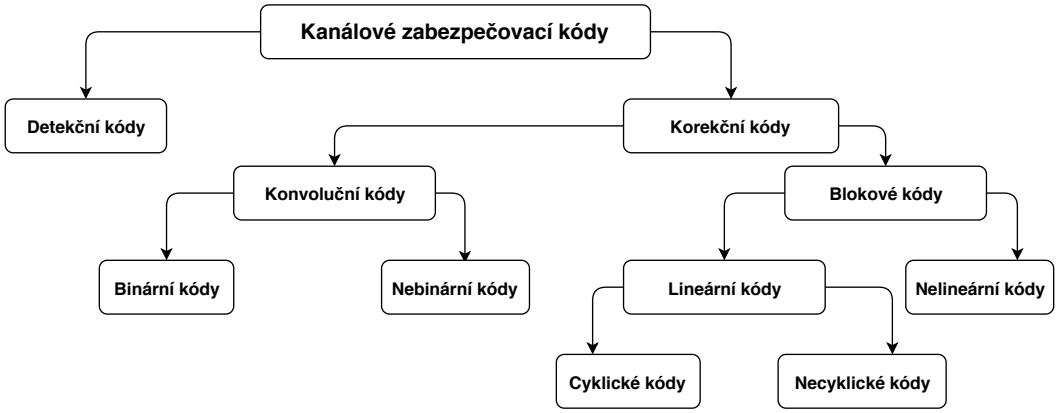
Na obrázku 1.2 je uvedeno dělení kanálových kódů. Zde se rozlišují kódy detekční a korekční. Detekční kódy jsou schopny chybu pouze odhalit, ale nikoli ji opravit. Tyto kódy jsou zpravidla jednodušší a méně náročné na výpočetní výkon, avšak vyžadují poloduplexní nebo duplexní přenos. Při detekci chyby je vyžádáno nové zaslání kódového slova. Pro korekční kódy není vyžadován zpětný přenosový kanál, stejně jako u detekčních kódů se nejdříve ověří, zda při přenosu nešlo ke změně přijatých bitů. Pokud je detekována chyba, pomocí algoritmu daného kódu dojde k pokusu o její opravu. Úspěšnost je závislá na tzv. Hammingově vzdálenosti d, která značí počet míst, v nichž se dvě značky liší, a taktéž na typu kódu. Minimální Hammingova vzdálenost $d_{\min}$ je definována pro množinu značek (kódovou abecedu) jako minimální počet míst, ve kterých se dvě libovolné značky daného kódu odlišují [1]. Z $d_{\min}$ lze podle (1.1) určit, kolik chyb je možno detekovat

$$k_d = d_{min} - 1. \tag{1.1}$$

Rovnice 1.2 říká, kolik chyb je teoreticky kód schopen opravit pro $d_{\min}$ liché a rovnice 1.3 pro $d_{\min}$ sudé

$$k_{ol} = \frac{d_{min} - 1}{2}, \tag{1.2}$$

$$k_{os} = \frac{d_{min} - 2}{2}. \tag{1.3}$$



Obr. 1.2: Dělení kanálových kódů [1].

1.1 Generující a kontrolní matice

Kanálové kódy se často označují jako kódy (n, k), kde k značí počet informačních bitů a n počet bitů zakódované zprávy. Z toho plyne, že při zabezpečení musí nutně docházet k určité redundanci. Z důvodu zabezpečujících schopností kódů nemůže kodér generovat množinu všech 2^n značek, protože jejich minimální Hammingova vzdálenost by byla jedna a zabezpečující vlastnosti kódu by byly nulové, ale pouze množinu 2^k odlišných značek délky n. K vytvoření kódových slov se často využívá tzv. generující matice, jejíž rozměr je $k \times n$ a obecný zápis pro generování systematického kódu je následovný:

$$G = [I_k P] = \begin{bmatrix} 1 & 0 & \dots & 0 & p_{11} & p_{12} & \dots & p_{1,(n-k)} \\ 0 & 1 & \dots & 0 & p_{21} & p_{22} & \dots & p_{2,(n-k)} \\ \vdots & & & \vdots & \vdots & & & \vdots \\ 0 & 0 & \dots & 1 & p_{k1} & p_{k2} & \dots & p_{k,(n-k)} \end{bmatrix}, \tag{1.4}$$

kde I_k je jednotková matice velikosti $k \times k$ a P je část matice zabezpečující kontrolu parity velikosti $(n-k) \times k$. [1] Postup kódování uvádí (1.5), kdy se vynásobením informačního slova m o délce k s generující maticí G získá kódové slovo c.

$$c = m.G \tag{1.5}$$

Při procesu kódování dochází k násobení informačního slova se sloupci generující matice. Jelikož pracujeme v binární logice, dochází ke sčítání bitů pomocí modulo(2) logiky.

Proces dekódování se skládá z několika kroků. Nejdříve se pomocí kontrolní matice H, která se získá odvozením z generující matice, vypočte tzv. syndrom s, jenž značí, zda při přenosu došlo k chybě či nikoli. Syndrom je dlouhý n-k bitů a pokud

jsou všechny bity nulové, k chybě nedošlo. Pokud je však minimálně jeden bit nenulový, došlo při přenosu k poškození zakódované zprávy a je nutno ji opravit. Proces opravy je podstatou kanálových kódů a bude pro jednotlivé kódy rozebrán v následujících kapitolách. Na závěr se ze zakódované zprávy zpětně dekódují informační bity.

$$s = c.H^T \tag{1.6}$$

Odvození kontrolní matice z matice generující je pro systematický blokový kód následující:

$$H = [-P^T I_{n-k}]. \tag{1.7}$$

Mezi generující a kontrolní maticí musí platit pravidlo ortogonality, tzn. že skalární součin dvou vektorových prostorů je nulový [6]. Díky tomu lze určit, zda přijaté slovo patří mezi platné kódové vektory odvozené z generující matice G.

$$G.H^T = 0 \tag{1.8}$$

1.2 Hammingův kód

Jedná se o nesystematický blokový kód se schopností opravy jednonásobných chyb [2]. V případě modifikace na $d_{min} = 4$ je navíc schopen detekovat dvojnásobnou chybu. Rozměry jednotlivých proměnných podle parametru p [3]:

Délka kódu: $n = 2^p - 1$
Počet informačních symbolů: $k = 2^p - p - 1$
Počet paritních symbolů: $n - k = p$

Podívejme se na nejjednodušší variantu (7,4), $k = 4$ informační bity, $n = 7$ bitů zakódované zprávy a $p = 3$ paritní bity. Pro tento případ bude velikost kontrolní matice H 3×7 bitů. Aby bylo možné jednoduše odhalit pozici chyby, je kontrolní matice sestavena tak, aby pořadí chyby odpovídalo dekadickému pořadí v zakódované zprávě.

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \tag{1.9}$$

Z kontrolní matice lze odvodit generující matici pomocí matematických úprav s přihlédnutím k (1.4) a (1.7). Nejdříve převedeme nesystematickou kontrolní matici na systematický tvar tak, aby na konci byla jednotková matice rozměru n-k. V tomto bodě lze odvodit generující matici systematického kódu podle (1.4), kdy na začátku je jednotková matice velikosti k a za ní je transponovaná část P z matice kontrolní.

Jelikož Hammingův kód je nesystematický, je v posledním kroku nutné přeskládat generující matici takovým způsobem, aby odpovídala tomuto systému. Aby byl kód jednoduše implementovatelný, paritní bity se vyskytují na pozicích, odpovídajících mocninám dvou. Informační bity jsou v Hammingově kódu zkopírovány na zbylé pozice. Toho je v generující matici dosaženo přeskládáním řádků tak, aby sloupce, ve kterých se vyskytuje právě jedna jednička, byly seřazeny od začátku matice sestupně, tzn. že první sloupec zleva s právě jednou jedničkou bude mít tuto jedničku umístěnou na první pozici odshora, další sloupec na druhé pozici odshora, atd.

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{1.10}$$

Kódování zprávy je v tomto okamžiku relativně jednoduché, stačí vynásobit informační bity m s generující maticí G podle (1.5). Jinou možností jak získat paritní bity, je vyjít ze skutečnosti, že tyto bity jsou na pozicích násobků mocnin dvou (1, 2, 4, 8, 16 ...), tudíž pro ně můžeme sestavit rovnice, vycházející z generující matice. Označme paritní bity proměnnou p a informační bity proměnnou m, potom bude vypadat zakódovaná zpráva následovně:

$$c = \begin{bmatrix} p_1 & p_2 & m_3 & p_4 & m_5 & m_6 & m_7 \end{bmatrix} \tag{1.11}$$

a z toho vyplývají tři algebraické rovnice pro paritní bity.

$$p_1 = m_3 + m_5 + m_7 \tag{1.12}$$

$$p_2 = m_3 + m_6 + m_7 \tag{1.13}$$

$$p_4 = m_5 + m_6 + m_7 \tag{1.14}$$

Zprávu lze tedy také zakódovat výpočtem paritních bitů pomocí těchto rovnic a naskládáním paritních a informačních bitů na správné pozice do kódového slova.

V tomto bodě máme zprávu zakódovanou. Po průchodu přenosovým kanálem je třeba ověřit, zda došlo k chybě přenosem. To zjistíme výpočtem syndromu podle (1.6). Pokud je syndrom nulový, přijatou zprávu není potřeba opravovat. V opačném případě přičteme jedničku k bitu na místo podle dekadické hodnoty syndromu. V posledním kroku dekódujeme z přijaté zprávy informační bity, které jsou na pozicích odpovídajících, buď sloupcům v generující matici obsahujících pouze jednu jedničku, nebo na místech v přijaté zprávě mimo mocniny dvou.

Celý proces kódování a dekódování si ukážeme na **příkladu**:
Budeme vycházet z nejjednodušší možnosti, tedy 3 paritní bity, 4 informační a celkově 7 bitů zakódované zprávy. Kód se potom značí jako HK(7,4). Informační bity zvolíme náhodně jako posloupnost $m = [1\ 0\ 1\ 1]$.

Kódujeme podle (1.5) a kódové slovo c získáme následovně:

$$c = m.G = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} . \tag{1.15}$$

Pokud bychom nyní vynásobili kódové slovo c s kontrolní maticí H, vyšel by syndrom nulový. Jelikož si však chceme ukázat, že je možno opravit jednonásobnou chybu, změníme v kódovém slově jeden bit jako simulaci průchodu přenosovým kanálem. Změňme například třetí bit zleva, potom bude přenesená zpráva vypadat následovně:

$$c_p = \begin{bmatrix} 0 & 1 & \mathbf{0} & 0 & 0 & 1 & 1 \end{bmatrix} . \tag{1.16}$$

Nyní provedeme výpočet syndromu pro kontrolu, zda v přenesené zprávě došlo k chybě, pokud ano, provedeme rovnou korekci:

$$s = c.H^T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} . \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} . \tag{1.17}$$

Syndrom odpovídá v dekadické soustavě číslu 3, což ukazuje správně na poškozený třetí bit. Nyní tedy stačí pouze přičíst ke třetímu bitu jedničku a zpráva je opravena.

$$c_k = \begin{bmatrix} 0 & 1 & \mathbf{0} & 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{1.18}$$

Posledním krokem je vyčtení informačních bitů ze zakódované zprávy. To je již relativně jednoduchý proces, kdy za sebe naskládáme bity na pozicích vyjma mocniny dvojky [0 1 **1** 1 0 **0 1 1**].

$$m = \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix} \tag{1.19}$$

1.3 Cyklický kód

Cyklické kódy se řadí mezi lineární blokové kódy. K jejich vytváření se často používají zpětnovazební posuvné registry. Výhoda je v jednoduchém sestavení kódovacího/ dekódovacího obvodu a dobré efektivnosti kódování.

Stejně jako u Hammingova kódu se rozměr cyklických kódu označuje (n, k), kde k značí počet informačních bitů a n počet bitů kódového slova. U cyklických kódů je obvyklé, že se informační slovo značí ve formě koeficientů polynomu (m_{k-1}, m_{k-2}, m_{k-3}, ..., m_0), se kterými se nám později lépe pracuje. Obdobně kódové slovo je značeno (c_{n-1}, c_{n-2}, c_{n-3}, ..., c_0) a pro tyto kódy je typické, že cyklický posuv doleva nebo doprava (c_{n-2}, c_{n-3}, c_{n-4}, ..., c_0, c_{n-1}) vytvoří opět platné kódové slovo z podprostoru kódových slov. Složky kódového slova jsou brány jako koeficienty mnohočlenu $c(x)$ a platí:

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1x^1 + c_0x^0 = \sum_{i=0}^{n-1} c_i x^i, \tag{1.20}$$

kde jednotlivé prvky c_i jsou vybírány z množiny konečných těles (ang. Galois Field) GF(2), kdy $c_i \in \ 0, 1$. Vysvětlení pojmu konečná tělesa bude věnována následující kapitola.

Princip kódování/dekódování vychází z generujícího polynomu $g(x)$, který musí splňovat určité požadavky:

- Stupeň mnohočlenu je $n - k$,
- Dělí beze zbytku dvojčlen $x^n - 1$,
- Pro opravu jednonásobných chyb musí být $g(x)$ alespoň trojčlenem a z množiny násobků $g(x)$ je nutno vyloučit všechny dvojčleny, aby minimální kódová vzdálenost byla $d_{min} > 2$ [4].

1.3.1 Konečná tělesa (GF)

Označení Galoisovo těleso je odvozeno podle jeho objevitele, který se jmenoval *Evarist Galois* (1811 - 1832). Galoisovo těleso obsahuje konečný počet prvků a jeho rozměr se vyjadřuje jako p^r. Kde r značí řád prvku konečného tělesa. Pro binární logiku je výhodné přiřadit $p = 2$, poté volba r určuje rozměr tělesa. Pro cyklické kódy je zásadní vytvoření Galoisova tělesa z generujícího polynomu $g(x)$, který musí splňovat pravidla uvedené výše. Ukažme si nyní na příkladu, jak vytvořit pole GF(16), tj. GF(2^4).

Příklad:

Vytvořme pole GF(2^4) s generujícím polynomem $g(x) = x^4 + x + 1$. Konstrukce vychází z předpokladu, kdy binární posloupnost odpovídá přepisu:

$0000 = 0$

$0001 = 1$

$0010 = x$

$0100 = x^2$

...

$1111 = x^3 + x^2 + x + 1$.

Prvky Galoisova tělesa se vypočítají jako zbytek po dělení generujícím polynomem $g(x)$. Tyto prvky se značí jako mocniny α a je jich celkem 2^4, tj. 0, α^0, α^1, α^2, α^3, ..., α^{14}. Další prvky jako α na vyšší mocniny nejsou pro konečná tělesa možné, avšak lze s nimi pracovat. Při násobení dochází ke sčítání exponentů a $\alpha^{15} = 1$, $\alpha^{16} = \alpha^{15} \,.\, \alpha = 1 \,.\, \alpha = \alpha$. Sčítání je také jednoduché, vyjádříme si dva prvky pod sebou rozepsané do binární logiky a jednotlivé bity sčítáme v logice modulo(2). Nyní si ukážeme samotné vytvoření Galoisových těles pomocí dělení mnohočlenu mnohočlenem. Nezajímá nás ani tak samotný výsledek, jako spíš zbytek po dělení. Ukažme si to například na:

$$\begin{array}{l} \alpha^7 \qquad\qquad\qquad \div \alpha^4 + \alpha + 1 = \alpha^3 + 1 \\ \underline{\alpha^7 + \alpha^4 + \alpha^3} \\ \quad \alpha^4 + \alpha^3 \\ \quad \underline{\alpha^4 + \alpha + 1} \\ \qquad \alpha^3 + \alpha + 1 \\ \qquad\qquad [1011] = x^3 + x + 1 \end{array}$$

Odtud je vidět, že prvku α^7 odpovídá prvek Galoisova pole $x^3 + x + 1$, což je v binárním zápisu 1011. Ze stejného principu vychází celá tabulka 1.1 o rozměru 16 prvků.

1.3.2 Nesystematické kódování

Princip kódování vychází z vytvoření generující matice G pomocí generujícího mnohočlenu. Ten se s každým řádkem přesune cyklickým posuvem o jedno místo doprava až jeho první člen dosáhne na poslední sloupec v generující matici. Proces kódování poté spočívá ve vynásobení informačních bitů (m_{k-1}, m_{k-2}, m_{k-3}, ..., m_0) se sloupci generující matice. Počet řádků matice odpovídá počtu informačních bitů.

Tab. 1.1: Prvky konečného tělesa GF(16).

α	bin	x
0	0000	0
1	0001	1
α	0010	x
α^2	0100	x^2
α^3	1000	x^3
α^4	0011	$x+1$
α^5	0110	x^2+x
α^6	1100	x^3+x^2
α^7	1011	x^3+x+1
α^8	0101	x^2+1
α^9	1010	x^3+x
α^{10}	0111	x^2+x+1
α^{11}	1110	x^3+x^2+x
α^{12}	1111	x^3+x^2+x+1
α^{13}	1101	x^3+x^2+1
α^{14}	1001	x^3+1

Výsledný vektor $c = [c_{n-1}, c_{n-2}, c_{n-3}, \ldots, c_0]$ poté značí kódové slovo.

$$c = [m_{k-1}, m_{k-2}, \ldots, m_0] \cdot \begin{bmatrix} g_{n-k} & g_{n-k-1} & \ldots & g_0 & 0 & 0 & \ldots & 0 \\ 0 & g_{n-k} & g_{n-k-1} & \ldots & g_0 & 0 & \ldots & 0 \\ \vdots & & & & & & & \\ 0 & 0 & \ldots & 0 & g_{n-k} & g_{n-k-1} & \ldots & g_0 \end{bmatrix} \tag{1.21}$$

Příklad:

Uvažujeme cyklický kód (7, 4) a pomocí generujícího polynomu $g(x) = x^3+x+1$ zakódujeme informační bity $m = [1\ 0\ 1\ 0]$.

V prvním kroku si z generujícího polynomu odvodíme generující matici:

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.22}$$

V kroku druhém provedeme vynásobení informačních bitů m s generující maticí G a po sečtení sloupců v modulo(2) logice zíkáme kódové slovo c.

$$c = m.G = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{1.23}$$

Na výsledku je vidět, že informační bity nejsou v kódovém slově zkopírovány, a proto je tohle kódování označováno jako nesystematické. V práci se však věnuji spíše cyklickým kódům systematickým, jejichž kódování je vysvětleno v následující kapitole.

1.3.3 Systematické kódování

Systematické kódování je možné provést několika způsoby. Nejdříve si ukážeme případ, kdy budeme vycházet z kontrolní matice, jako tomu bylo u Hammingova kódování. Kontrolní matici sestavíme sestupně z prvků α konečného tělesa. Díky tomu budeme schopni jednoduše odhalit jednonásobnou chybu a provést její korekci. Kontrolní matice má tedy následující tvar:

$$H = \begin{bmatrix} \alpha^{14} & \alpha^{13} & \alpha^{12} & \alpha^{11} & \alpha^{10} & \alpha^{9} & \alpha^{8} & \alpha^{7} & \alpha^{6} & \alpha^{5} & \alpha^{4} & \alpha^{3} & \alpha^{2} & \alpha & 1 \end{bmatrix}, \tag{1.24}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.25}$$

Generující matice je poté sestavena tak, že na začátku je jednotková matice o rozměru $k \times k$ a na konci je transponovaná H matice o počtu k sloupců.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{1.26}$$

V tomto okamžiku máme k dispozici generující a kontrolní matici a můžeme si ukázat, jak funguje proces kódování. Pro jednoduchost budeme kódovat pouze 4 informační bity.

Příklad:

Pomocí systematického cyklického kódu (7, 4) zakódujeme informační bity $m = [1\ 0\ 1\ 0]$. Využijeme generující polynom $g(x) = x^3 + x + 1$.

Tab. 1.2: Prvky konečného tělesa GF(8).

α	bin	x
0	000	0
1	001	1
α	010	x
α^2	100	x^2
α^3	011	$x+1$
α^4	110	x^2+x
α^5	111	x^2+x+1
α^6	101	x^2+1

Odvozená kontrolní H a generující G matice mají tvar:

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \tag{1.27}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.28}$$

Proces kódování je dle (1.5) následující:

$$c = m.G = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{1.29}$$

Zakódované slovo $c = [1\ 0\ 1\ 0\ 0\ 1\ 1]$. Jak je vidět, tak na začátku zleva, což jsou nejvýznamnější bity, jsou zkopírovány 4 informační bity a po nich následují 3 bity kontrolní.

Jelikož je však u kódů vyšších rozměrů nutné počítat velké množství prvků Galoisova pole, není kódování pomocí generující matice, která vychází z matice kontrolní, úplně výhodné. Ukážeme si, jak efektivně kódovat informační bity pomocí generujícího mnohočlenu. Tento způsob kódování vychází z dělení mnohočlenu mnohočlenem. Informační bity ($m_{k-1}\ x^{k-1}$, $m_{k-2}\ x^{k-2}$, $m_{k-3}\ x^{k-3}$, ..., $m_0\ x^0$) vynásobíme prvkem x^{n-k}, čímž dojde k doplnění slova zprava nulami, neboli informační slovo se posune doleva o n-k pozic. Takto nově vzniklé slovo označme např. $z(x)$. Poté dělíme takto posunuté informační slovo generujícím polynomem $g(x)$. Tím dostaneme podíl $q(x)$ a nedělitelný zbytek $r(x)$ stupně n-k-1 nebo nižšího. Kódové slovo $c(x)$ vytvoříme součtem $z(x)$ a zbytkem po dělení $r(x)$.

Příklad:

Pomocí systematického cyklického kódu (7, 4) zakódujeme informační bity $m = [1\ 0\ 1\ 0]$. Využijeme generující polynom $g(x) = x^3 + x + 1$.

Informační bity vynásobíme prvkem x^{n-1}.

$$z(x) = m.x^{n-k} = (x^3 + x).x^3 = x^6 + x^4 \tag{1.30}$$

Nyní dělíme slovo $z(x)$ generujícím mnohočlenem $g(x)$. Tím dostaneme podíl $q(x)$ a nedělitelný zbytek $r(x)$ stupně n-k-1 nebo nižšího.

$$\frac{z(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \tag{1.31}$$

$$\begin{array}{l} x^6 + x^4 \qquad\qquad \div\, x^3 + x + 1 = x^3 + 1 \\ \underline{x^6 + x^4 + x^3} \\ \quad x^3 \\ \quad \underline{x^3 + x + 1} \\ \qquad x + 1 = [0\ 1\ 1] = r(x) \end{array}$$

V posledním kroku sečteme informační bity $z(x)$ a zbytek po dělení $r(x)$ a dostaneme kódové slovo cyklického systematického kódu.

$$c(x) = z(x) + r(x) = \left[\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right] + \left[\begin{array}{ccc} 0 & 1 & 1 \end{array}\right] = \left[\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \tag{1.32}$$

Můžeme porovnat s předchozím příkladem a je jasně vidět, že kódové slovo je v obou případech stejné.

Nyní umíme informační bity kódovat pomocí cyklického kódu a můžeme se podívat na dekódování a případnou opravu chyby vzniklou přenosem komunikačním kanálem. Nejjednodušší bude opět ukázat princip na příkladech. Začněme dekódováním s pomocí kontrolní matice H.

Příklad:

Uvažujme, že bylo vysláno slovo z příkladu výše, tedy $c(x) = [1\ 0\ 1\ 0\ 0\ 1\ 1]$. Při přenosu došlo k chybě na páté pozici zprava a bylo přijato kódové slovo $[\ 1\ 0\ \mathbf{0}\ 0\ 0\ 1\ 1]$. Rozhodněme, zda při přenosu došlo k chybě a případně ji opravme.

Zda se jedná o platné kódové slovo, ověříme vynásobením přijaté posloupnosti s kontrolní transponovanou H maticí. Pakliže je syndrom nulový, můžeme prohlásit, že přenos proběhl bez chyby. V opačném případě se pokusíme o opravu.

$$s = c.H^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \tag{1.33}$$

Syndrom není nulový, je jasné, že při přenosu došlo k chybě. Pozici změněného bitu je možné určit porovnáním syndromu s řádky matice H^T. Syndrom se v matici shoduje s pátým řádkem odspodu, což odpovídá chybně přenesenému bitu na páté pozici zprava. Provedeme tedy negaci tohoto bitu a získáme platné kódové slovo $c = [1\ 0\ \mathbf{1}\ 0\ 0\ 1\ 1]$. Informační bity jsou potom na pozicích čtyř nejvýznamnějších bitů, tj. $m = [1\ 0\ 1\ 0]$.

Druhý postup dekódování vychází z dělení přijatého kódového slova generujícím polynomem. Pokud je zbytek po dělení nulový, bylo pravděpodobně přijato správné kódové slovo. Pokud není, můžeme se opět pokusit o opravu. Viz následující příklad.

Příklad:

Uvažujme, že bylo vysláno slovo z příkladu výše, tedy $c(x) = [1\ 0\ 1\ 0\ 0\ 1\ 1]$. Při přenosu došlo k chybě na páté pozici zprava, a bylo tedy přijato kódové slovo $[\ 1\ 0\ \mathbf{0}\ 0\ 0\ 1\ 1]$. Rozhodněme, zda při přenosu došlo k chybě a případně ji opravme.

Kontrolu, zda došlo při přenosu k chybě, provedeme vydělením přijatého slova generujícím polynomem $g(x)$.

$$x^6 + x + 1 \qquad \div x^3 + x + 1 = x^3 + x + 1$$
$$x^6 + x^4 + x^3$$
$$\downarrow$$
$$x^2 + x = [1\ 1\ 0]$$

Na první pohled je vidět, že syndrom je stejný jako v předchozím příkladě. Nyní ovšem nemáme k dispozici tabulku syndromů. Postup je následovný:

- Pokud je Hammingova váha syndromu rovna jedné, a jelikož kód umí opravovat pouze jednonásobnou chybu, opravu provedeme přičtením syndromu k přijaté kódové posloupnosti.
- Je-li Hammingova váha větší než jedna, budeme postupně provádět cyklické posuvy prvků ve značce a znovu počítat syndromy tak dlouho, dokud nebude Hammingova váha rovna jedné. Poté sečteme kódové slovo se syndromem a provedeme stejný počet posuvů zpět. Nezáleží, kterým směrem provádíme cyklické posuvy, ale musíme se držet jednoho směru a na konci posunout slovo správným směrem zpět.

Jelikož je Hammingova váha syndromu rovná 2, posuneme cyklicky kódové slovo vpravo a znovu vypočteme hodnotu syndromu:

$$x^6 + x^5 + 1 \qquad \div x^3 + x + 1 = x^3 + x^2 + x$$
$$x^6 + x^4 + x^3$$
$$\downarrow$$
$$x + 1 = [0\ 1\ 1]$$

Hammingova váha syndromu je stále 2, znovu posuneme vpravo a počítáme syndrom:

$$x^6 + x^5 + x^4 \qquad \div x^3 + x + 1 = x^3 + x^2 + x$$
$$x^6 + x^4 + x^3$$
$$\downarrow$$
$$x^2 = [1\ 0\ 0]$$

Po dvou posunech vpravo je Hammingova váha syndromu rovna jedné. Můžeme tedy ke kódovému slovu syndrom přičíst a provést dva posuvy zpět, díky čemuž slovo opravíme.

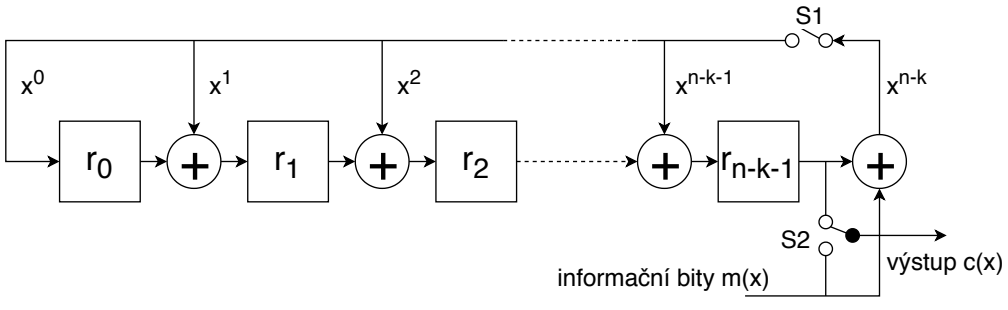
$$\begin{array}{r} 1\,1\,1\,0\,0\,0\,0 \\ \qquad\quad 1\,0\,0 \\ \hline 1\,1\,1\,0\,1\,0\,0 \rightarrow\rightarrow 1\,0\,1\,0\,0\,1\,1 \end{array}$$

Nyní kódové slovo opravdu odpovídá vyslanému a je možno vyčíst informační bity ze čtyřech pozic nejvíce významných bitů, $m = [1\ 0\ 1\ 0]$.

1.3.4 Realizace kodéru pomocí zpětnovazebních posuvných registrů

Jak bylo uvedeno výše, výhoda použití cyklických kódu je v jejich snadné realizaci pomocí posuvných registrů. V dnešní době rychlých a dostupných mikroprocesorů přestává tato výhoda hrát ve prospěch cyklických kódů, ale přesto jsou stále běžně používané. Ukážeme si proto, jak realizace pomocí posuvných registrů funguje.

Nejdříve se podívejme na princip kódování systematického cyklického kódu. Na obrázku 1.3 je znázorněno obecné blokové schéma s využitím posuvných registrů a sčítaček modulo(2). Postup je následující:



Obr. 1.3: Obecné blokové schéma kodéru CK s posuvnými registry.

- Počet posuvných registrů odpovídá stupni generujícího mnohočlenu $g(x)$, přičemž platí, že členy x^0 a x^{n-k} jsou v $g(x)$ vždy obsaženy.
- Zda je mezi posuvnými registry sčítací člen, záleží na uspořádání $g(x)$, konkrétně na tom, jestli je daný člen v polynomu obsažen, viz obrázek 1.3, kde jsou pozice jednotlivých členů vyznačeny.
- Informační bity jsou pomocí zpětné vazby přiváděny na začátek pole posuvných registrů, přičemž spínač S1 je sepnut, dokud nejsou všechny informační bity nasunuty.
- Po nasunutí posledního informačního bitu dojde k rozpojení spínače S1 a k přepnutí spínače S2.
- Spínač S2 slouží k oddělení informačních a kontrolních bitů. V první fázi jdou informační bity zároveň do posuvných registrů, a také se ukládají v paměti.
- Po nasunutí posledního informačního bitu je v registrech uložen zbytek po dělení, který má doplnit informační bity. Rozpojením spínače S1 s přepnutím S2 dochází v hodinovém taktu k jejich postupnému vyčítání, kdy na konci posledního cyklu dostaneme celé kódové slovo.
- Všechny registry mají v tomto okamžiku hodnotu 0 a může se začít kódovat nové slovo.

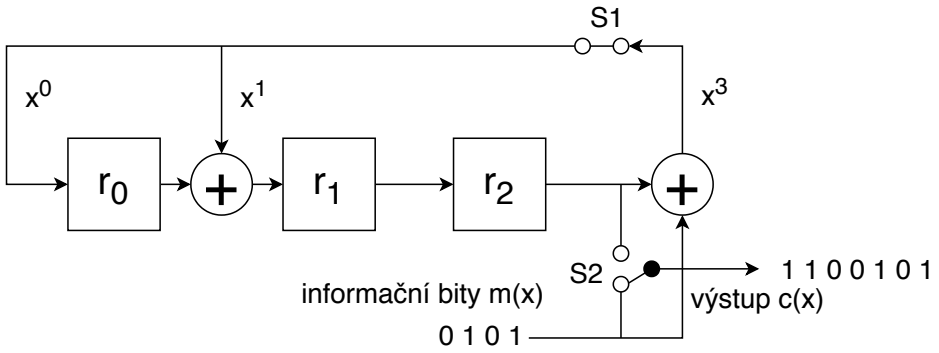
Popsaný postup si nyní předvedeme na příkladu. Pro kontrolu, abychom ukázali, že je výsledek stejný jako v předešlých postupech, budeme kódovat tytéž informační bity.

Příklad:

Pomocí systematického cyklického kódu (7, 4) zakódujeme informační bity $m = [1\ 0\ 1\ 0]$. Využijeme generující polynom $g(x) = x^3 + x + 1$.

Podle výše zmíněného postupu vytvoříme schéma kodéru. Prvek s nejvyšší mocninou v $g(x)$ je x^3, takže budeme mít 3 posuvné registry. Prvek polynomu x^0 značí zpětnou vazbu a je obsažen vždy. Podle prvku x^1 bude sčítačka mezi registry r_0 a r_1, viz obrázek 1.4.

Do kodéru se informační bity nasouvají od prvku s nejvyšší váhou. Potom je na výstupu kódové slovo seřazeno v opačném pořadí, zprava jsou umístěny informační bity a následují bity kontrolní. Jak se kontrolní bity počítají, a jak vzniká celé kódové slovo, ukazuje tabulka 1.3. Kontrolní bity je také možné po nasunutí posledního informačního bitu vyčíst z registrů paralelně.



Obr. 1.4: Blokové schéma kodéru CK s posuvnými registry.

Tab. 1.3: Průběh kódování CK s posuvnými registry.

krok	vstup (inf. bity)	r_0	r_1	r_2	výstup	stav spínačů
0	-	0	0	0		S1 sepnut, S2 spodní pozice
1	1	1	1	0	**1**	S1 sepnut, S2 spodní pozice
2	0	0	1	1	**0**	S1 sepnut, S2 spodní pozice
3	1	0	0	1	**1**	S1 sepnut, S2 spodní pozice
4	0	1	1	0	**0**	S1 sepnut, S2 spodní pozice
5		0	1	1	**0**	S1 rozepnut, S2 horní pozice
6		0	0	1	**1**	S1 rozepnut, S2 horní pozice
7		0	0	0	**1**	S1 rozepnut, S2 horní pozice

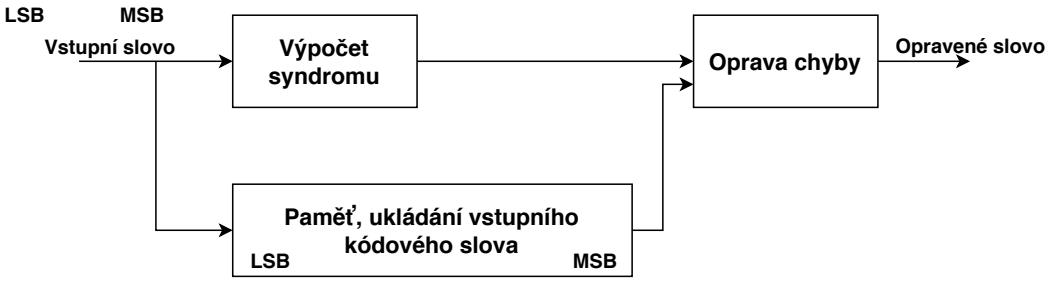
1.3.5 Realizace dekodéru pomocí zpětnovazebních posuvných registrů

Proces opačný ke kódování se nazývá dekódování. Při dekódování, stejně jako v předchozích odstavcích, kontrolujeme, zda nedošlo k chybě při přenosu a případně vyskytující se chybu v rámci schopností kódu opravíme. Zmíněnou operaci lze také realizovat pomocí posuvných registrů a logických obvodů. Blokové schéma dekodéru je rozděleno do tří funkčních celků:

- Blok pro výpočet syndromu,
- Blok posuvných registrů pro ukládání vstupního kódového slova,
- Blok pro opravu chyby.

Dekódování probíhá následovně:

- Nejdříve se stejně jako při kódování vypočte syndrom. Ten je kompletní po nasunutí celého kódového slova, které se opět nasouvá od nejvíce významného bitu.
- Zároveň se přijaté kódové slovo ukládá do paměti posuvných registrů.



Obr. 1.5: Blokové schéma obecného dekodéru CK s posuvnými registry.

- Pokud je po nasunutí všech bitů syndrom nulový, znamená to, že slovo nebylo při přenosu poškozeno a není třeba jej opravovat.
- Není-li syndrom nulový, slovo je poškozeno a bude potřeba jej opravit. K tomu slouží blok oprava chyby, ve kterém se modulo(2) sčítají bity přijatého kódového slova a nově vypočteného korekčního bitu.

Určení korekčního bitu je následující a bude blíže předvedeno na příkladu:

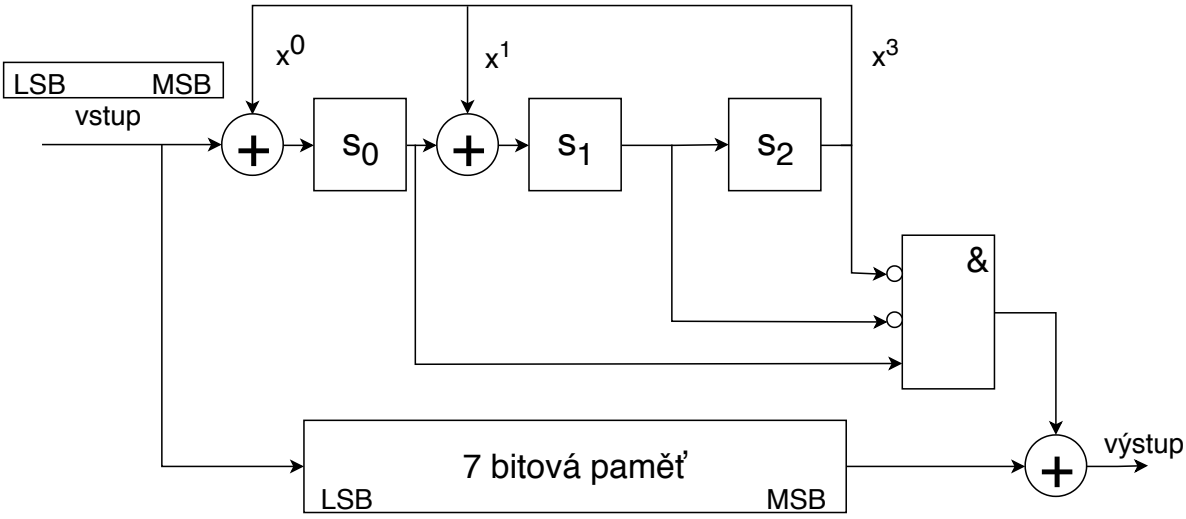
- Po dokončení výpočtu syndromu jsou v registrech uloženy bity syndromu a v paměti kompletní přijaté kódové slovo.
- Jinou možností jak opravit chybu, by bylo nahlédnutí do tabulky syndromů viz např. tabulka 1.2. Při použití posuvných registrů však tabulka není k dispozici, proto budeme provádět výpočet opravného bitu.
- V dalších n krocích je na vstup přiváděn bit "0", čímž se přepočítává syndrom pro opravu chyby.
- S každým i-tým novým syndromem zjistíme, zda není chybný právě bit n-i a oprava může být ukončena dříve.
- Blok opravy chyby musí být uspořádán tak, aby v případě chybného prvního bitu přijaté zprávy, tj. chybový vektor [0 0 0 0 0 0 1] x^0, čemuž odpovídá syndrom [0 0 1] x^0, byl vygenerován opravný bit "1".

Příklad:

Uvažujme, že bylo vysláno slovo z příkladu výše, tedy $c(x) = [1\ 0\ 1\ 0\ 0\ 1\ 1]$. Při přenosu došlo k chybě na páté pozici zprava, a bylo tedy přijato kódové slovo [1 0 **0** 0 0 1 1]. Rozhodněme, zda při přenosu došlo k chybě a případně ji opravme.

K detekci a opravě chyby sestavíme dekodér viz obrázek 1.6. Dále dle postupu uvedeného výše vypočteme syndrom a pokud nebude nulový, budeme pokračovat a slovo opravíme. Kroky výpočtu syndromu jsou uvedeny v tabulce 1.4.

Jelikož syndrom není nulový, je v přijatém slově chyba. Aplikací kroků uvedených výše chybu opravíme. Nyní máme v paměti celé přijaté slovo napadené chybou



Obr. 1.6: Blokové schéma dekodéru CK s posuvnými registry.

Tab. 1.4: Výpočet syndromu CK.

krok	vstup (kód. slovo)	s_0	s_1	s_2
0	-	0	0	0
1	1	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	1	1	0
5	0	0	1	1
6	1	0	1	1
7	1	**0**	**1**	**1**

a v registrech hodnotu syndromu. Na vstup přivedeme sedmibitový nulový vektor, budeme postupně přepočítávat syndrom v registrech, pomocí logického obvodu AND počítat opravný bit a v logice modulo(2) ho sčítat s bity přijatého kódového slova. Jak je vidět dále v tabulce 1.5, opravný bit je hodnoty "1"pouze pro chybný bit v přenesené zprávě, čímž dojde k opravě chyby.

Jak je patrno z tabulky 1.5 chyba byla opravena po 3 krocích, tudíž by mohla být oprava ukončena dříve. Z toho plyne, že rychlost opravy záleží na pozici chyby ve zprávě. Na čím významnější pozici chyba je, tím dříve je opravena. Posledním krokem je vyčtení informačních bitů. Jelikož víme, že kódové slovo je v opačném pořadí, můžeme jednoduše určit, že $m = [1\ 0\ 1\ 0]$.

Tab. 1.5: Oprava chyby CK.

krok	vstup	s_0	s_1	s_2	opravný bit	kódové slovo	opravené slovo
0	-	0	1	1			
1	0	1	1	1	0	1	1
2	0	1	0	1	0	0	0
3	0	1	0	0	**1**	0	**1**
4	0	0	1	0	0	0	0
5	0	0	0	1	0	0	0
6	0	1	1	0	0	1	1
7	0	0	1	1	0	1	1

1.4 Konvoluční kód

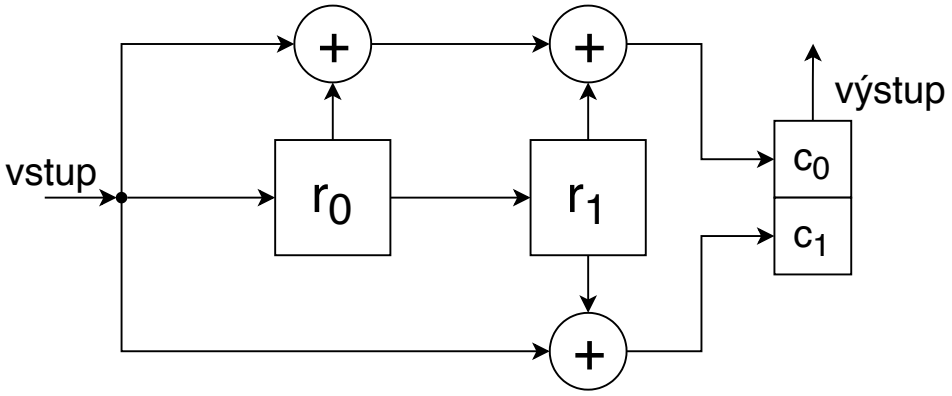
Doposud popisované kódy spadaly do kategorie blokových viz obr 1.2. To znamená, že daný kód se aplikoval na přesně daný počet informačních bitů, přičemž se označovaly jako kódy bez paměti. Na rozdíl od toho je konvoluční kód označován jako kód s pamětí, jelikož výstupní hodnota nezávisí pouze na vstupním bitu, ale také na několika předchozích, to záleží na kódové délce. Podle toho se opět označují jako (n, k) kódy, kde n značí, kolik bitů bude na výstupu z k bitů vstupních. Pokud dáme tyto hodnoty do poměru, získáme tzv. *rychlost kódu* (R).

$$R = \frac{k}{n} \tag{1.34}$$

1.4.1 Kódování

Kódování vychází z kódovacího blokového kodéru, jehož možné schéma je na obrázku 1.7, a z něhož lze odvodit generující polynomy [5], případně impulzní odezvu [8]. Další možností, jak zakódovat informační slovo, je použití kódovací tabulky nebo kódového stromu [13]. Nejčastější a pro názornou ukázku jako nejlepší se jeví možnost použití mřížového (trellis) diagramu.

Jak je vidět z kodéru na obr 1.7, jedná se o kód (2, 1), jelikož kodér z jednoho informačního bitu generuje dva bity výstupní. Funkce kodéru je následovná. Na počátku jsou paměťové buňky (posuvné registry) vynulovány, takže na výstupu jsou taktéž nuly. Přivedeme-li na vstup posloupnost $m = [1\ 0\ 1\ 0]$, budou informační bity postupovat kodérem a na výstupu se budou postupně objevovat dvojice kódových bitů. Z toho vyplývá, že kódování probíhá průběžně s tokem informace. Přiveďme na vstup první informační bit 1, podle zapojení kodéru dojde v horní větvi ke dvěma modulo(2) sečtením s obsahy obou posuvných registrů. Ve spodní větvi se informační



Obr. 1.7: Blokové schéma kodéru konvolučního kódu.

bit sečte pouze s obsahem druhého registru. Na výstupu se tedy objeví dvojice kódových bitů s hodnotami 11. Průběh kódování ukazuje tabulka 1.6. Když kódové bity seřadíme za sebe, dostaneme kódové slovo $c = [1\ 1\ 1\ 0\ 0\ 0\ 1\ 0]$.

Tab. 1.6: Průběh kódování konvolučního kódu.

krok	vstup	registr r_0	registr r_1	výstup c_0	výstup c_1
0	0	0	0	0	0
1	1	0	0	1	1
2	0	1	0	1	0
3	1	0	1	0	0
4	0	1	0	1	0

Jelikož u lineárního konvolučního kódu platí princip superpozice, je také možné kódování chápat jako násobení mnohočlenů. Kodér z obrázku 1.7 má dva generující mnohočleny $g(x1) = x^2 + x + 1$ a $g(x2) = x^2 + 1$. Poté je možné zprávu zakódovat následujícím způsobem [1], [5] :

Příklad:

Pomocí konvolučního kódu a jeho generujících polynomů $g(x1) = x^2 + x + 1$ a $g(x2) = x^2 + 1$ zakódujte informační slovo $m = [1\ 0\ 1\ 0]$.

Informační slovo m převedeme do formy mnohočlenu jako $m = x^3 + x$. Nyní provedeme násobení každého generujícího polynomu s informačním slovem a výsledky budou tvořit jednotlivé kódové slova c_0 a c_1. Dvojice bitů poskládáme k sobě a dostaneme celé kódové slovo.

$$c_0 = g(x1).m = (x^2 + x + 1).(x^3 + x) = x^5 + x^4 + x^2 + x \tag{1.35}$$

$$c_1 = g(x2).m = (x^2 + 1).(x^3 + x) = x^5 + x \tag{1.36}$$

Převedeme výsledky násobení do binární logiky, takže:
$c_0 = [1\ 1\ 0\ 1\ 1]$ a $c_1 = [1\ 0\ 0\ 0\ 1]$. V posledním kroku k sobě seřadíme jednotlivé bity a kódové slovo $c = [11\ 10\ 00\ 10\ 11]$. Poslední dvojice bitů na konci je pouze "doběh" do nulové polohy v mřížovém diagramu a v literatuře se značí jako *tail of the message* [4]. Výsledek se shoduje s postupem výše.

Přivedením jednotkového impulsu na vstup kodéru získáme impulsní odezvu [1], neboli generující mnohočlen. Tento proces lze provést také obráceně a z impulzní odezvy odvodit zpětně schéma kodéru. Možné uplatnění se nachází při sestavení generující matice. Ukažme si na příkladu, jak impulzní odezvu získat:

Příklad:

Přivedením jednotkového impulzu na vstup kodéru z obrázku 1.7 odvoďme jeho impulzní odezvu.

Impulzní odezva je složena z výstupních hodnot kodéru, dokud se stav paměťových buněk (posuvných registrů) kodéru neustálí na nulu, tzn. dokud vstupní jednička neprojde posledním posuvným registrem. Postup určení impulzní odezvy je v tabulce 1.7. Impulzní odezva je poté [1 1 1 0 1 1].

Získanou impulzní odezvu je možné využít ke kódování informačního slova sestavením generující matice. Stejně jako u předchozích kódů se impulzní odezva vpisuje do řádků. Zde však záleží na rozměrech kódu, tj. z kolika k informačních bitů se generuje n bitů kódových. Podle toho dochází v každém řádku k posunutí o n míst.

Tab. 1.7: Určení impulzní odezvy.

krok	vstup	registr r_0	registr r_1	výstup c_0	výstup c_1
0	0	0	0	0	0
1	1	0	0	1	1
2	0	1	0	1	0
3	0	0	1	1	1

$$G = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & \dots \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & \dots & \dots \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots \end{bmatrix} \tag{1.37}$$

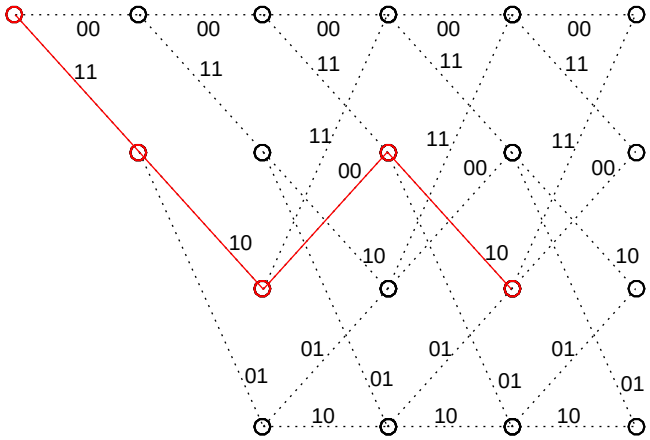
Vzorová matice není konečná, jelikož záleží na počtu informačních bitů, které chceme kódovat. Proto je naznačeno pokračování do všech směrů. Proces kódování dále vychází z rovnice (1.5).

Další možností, jak zakódovat informační bity, je s pomocí kódového stromu [7], který je zobrazen na obrázku 1.9. Jedná se o diagram, jenž zachycuje všechny možné stavy registrů a podle hodnoty bitu na vstupu ukazuje, jaká posloupnost bitů se zobrazí na výstupu. Pokud je na vstupu hodnota "1", postupuje se v diagramu směrem nahoru, je-li na vstupu bit "0", přechází se směrem dolů. Hodnoty výstupních bitů jsou pro každý krok zapsány na vodorovných linkách. Pro větší množství bitů na vstupu však roste diagram do velkých a nepřehledných rozměrů. Proto se přechází na mřížový (trellis) diagram, který má sloučené stejné stavy a jeho podoba je zobrazena na obrázku 1.10. V prvním kroku, když jsou hodnoty registrů nulové, jsou další možné stavy pouze dva, poté čtyři a od třetího kroku je vždy osm možných stavů registrů. Podle hodnoty vstupního bitu se opět pohybujeme v mřížovém diagramu. Při nulovém vstupním bitu se pohneme horní větví a při hodnotě vstupního bitu "1" jdeme spodní větví. Aktuální stavy paměťových registrů, ze kterých se vychází, jsou zapsány na začátku každého řádku, a tyto stavy jsou neměnné. Hodnoty výstupních bitů jsou zapsány k jednotlivým propojům mezi uzly. Na obrázku 1.11 je navíc zobrazen stavový diagram. Postup kódování pomocí mřížového diagramu si předvedeme na následujícím příkladu:

Příklad:

Pomocí mřížového (trellis) diagramu a konvolučního kódu (2, 1) s generujícím mnohočlenem $g(x) = x^5 + x^4 + x^3 + x + 1$ zakódujte informační slovo $m = [1\ 0\ 1\ 0]$.

Mřížový diagram, který lze jednoduše odvodit, odpovídá diagramu z obrázku 1.10. Kódování probíhá podle pravidel popsaných výše a cesta pro dané informační slovo je zvýrazněna na obrázku 1.8. Poté lze odečíst kódové bity, jejichž poskládáním za sebe vznikne hledané kódové slovo c = [1 1 1 0 0 0 1 0 1 1].



Obr. 1.8: Mřížový (trellis) diagram se zvýrazněním cesty pro kódování.

1.4.2 Dekódování

Dekódování probíhá nejčastěji tak, že se porovná přijatá zpráva, která může být zasažena chybou se všemi posloupnostmi, jež mohly být vyslány. Hledá se tzv. věrohodnostní poměr a snažíme se o jeho maximalizaci. Pokud je přijetí všech kódových slov stejně pravděpodobné, pak dekodér, který dosahuje nejlepšího věrohodnostního poměru, je ten, který porovnává podmíněné pravděpodobnosti. Také je možné vyjádřit pravděpodobnostní funkci viz (1.38). Kde Z značí přijaté slovo a U^m některou z možných přijatých sekvencí. Dekodér vybere $U^{m\prime}$, jestliže [8]

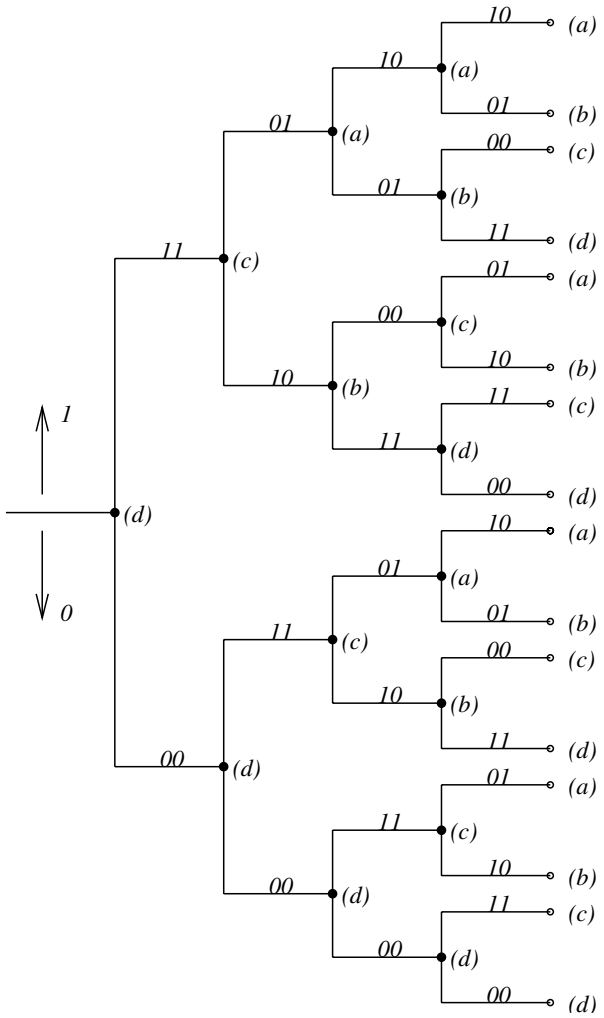
$$P(Z|U^{m'}) = maxP(Z|U^m)/\ přes\ všechny\ U^m. \tag{1.38}$$

Předpokládáme-li kanál s aditivním Gaussovským šumem s nulovou střední hodnotou a, také kanál bez paměti, tzn. že šum ovlivní každý symbol nezávisle na ostatních symbolech. Pak pro konvoluční kód s rychlostí $1/n$ můžeme napsat věrohodnost jako [8]:

$$P(Z|U^m) = \prod_{i=1}^{\infty} P(Z_i|U_i^m), \tag{1.39}$$

kde Z_i je i-tá větev přijaté sekvence Z, U_i^m je i-tá větev konkrétní kódové sekvence U^m, z_{ji} je j-tý kódový symbol Z_i a u_{ji}^m je j-tý kódový symbol U_i^m. Každá větev se skládá z n kódových symbolů [8].

Dekodér tedy musí najít cestu kódovým stromem nebo mřížovým diagramem tak, aby pravděpodobnost (1.39) byla maximalizována. Je-li přijatá binární sekvence Z vytvořena z L slov definovaných větvemi kódu, pak počet posloupností U^m , se kterými musí být přijatá posloupnost porovnána, je 2^L [1].

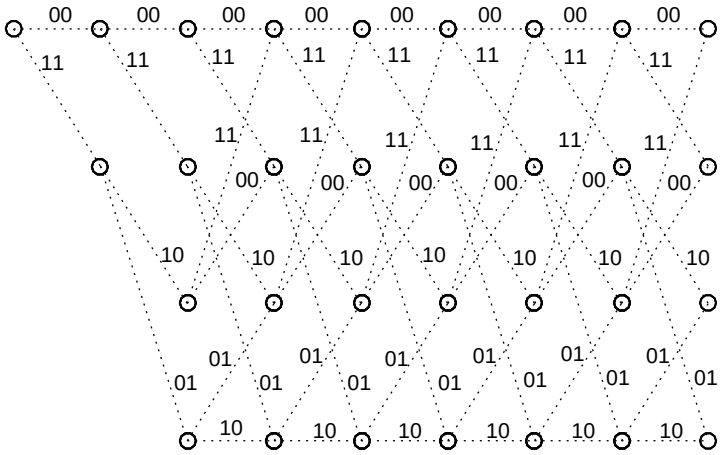


Obr. 1.9: Schéma kódového stromu pro konvoluční kód [7].

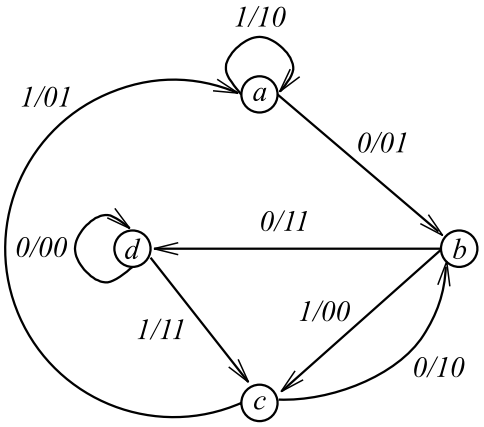
V roce 1967 *A. J. Viterbi* vyvinul velice efektivní metodu dekódování založenou na maximalizaci věrohodnostního poměru [15]. Dekódování spočívá v porovnání přijaté zprávy se všemi možnými cestami v mřížovém diagramu. Zároveň se pro jednotlivé cesty počítá Hammingova vzdálenost, tj. rozdíl mezi přijatým a možným kódovým slovem. Jako nejpravděpodobnější kódové slovo se nakonec určí to, jehož cesta mřížovým diagramem má nejmenší celkovou Hammingovu vzdálenost od přijatého slova. Pokud je celková Hammingova vzdálenost rovná nule, bylo pravděpodobně přijato právě vyslané kódové slovo. Postup dekódování Viterbiho algoritmem si nyní ukážeme na příkladu.

Příklad:

Pomocí Viterbiho algoritmu a mřížového diagramu na obrázku 1.10 dekódujte informační bity z přijatého kódového slova $c = [1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1]$.



Obr. 1.10: Mřížový (trellis) diagram.



Obr. 1.11: Stavový diagram [7].

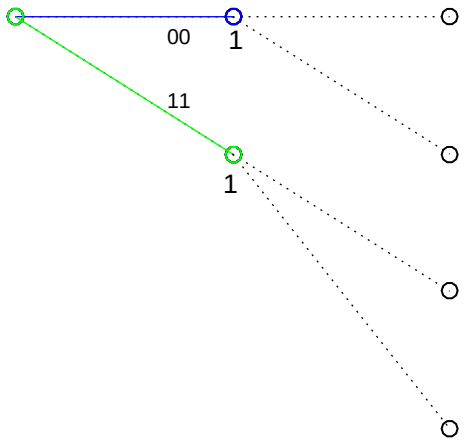
Pro úplnost uveďme, že při přenosu došlo k dvěma chybám a skutečně vyslané kódové slovo bylo [1 **1** 1 0 0 **0** 1 0 1 1] (chybně přenesené bity jsou označeny tučně), přičemž informační slovo odpovídá $m = [1\ 0\ 1\ 0\ 0]$. Postup dekódování je následující:

- Nejdříve se přijatá zpráva rozdělí na bloky po n bitech, kdy každý blok vyjadřuje jeden informační bit.
- Poté se bloky rozepíší nad jednotlivé úseky v mřížovém diagramu a porovnávají se se všemi možnými cestami, přičemž se zaznamenává vzájemná Hammingova váha, tj. počet odlišností, které se iterativně sčítají.
- V prvním úseku jsou pouze dvě možné cesty, v druhém čtyři a v každém dalším osm.
- Jelikož se v každém dalším kroku sejdou v jednom ze čtyř uzlů dvě možné

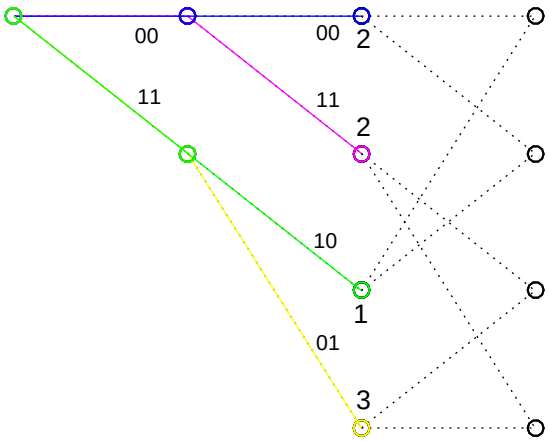
cesty, vybere a zachová se z nich pouze ta, která má menší Hammingovu vzdálenost vůči přijaté zprávě. Pokud by měly dvě cesty stejnou Hammingovu váhu, vybere se náhodně jedna z nich.

- Na konci dekódování se určí jako nejpravděpodobnější cesta (vyslané slovo) ta, která má nejmenší celkovou Hammingovu váhu.
- Pokud by došlo k případu, což se může stát, že na konci dekódování mají dvě nebo více cest stejnou Hammingovu váhu, nelze jednoznačně určit vyslané slovo.

Na obrázcích 1.12 až 1.16 jsou vyobrazeny jednotlivé dekódovací kroky a k jednotlivým cestám zaznamenána Hammingova vzdálenost vůči přijatému slovu.

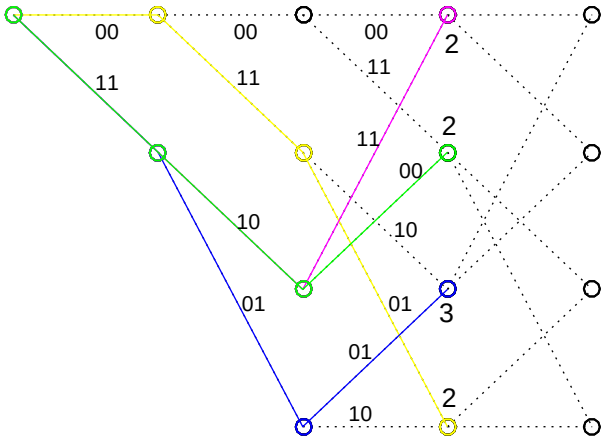


Obr. 1.12: Dekódování pomocí Viterbiho algoritmu 1. krok.

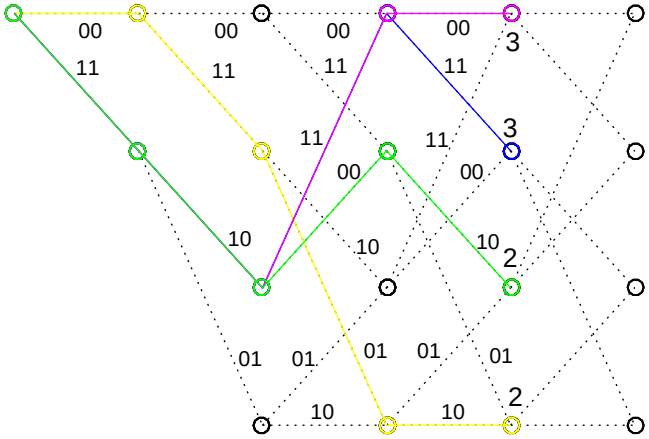


Obr. 1.13: Dekódování pomocí Viterbiho algoritmu 2. krok.

Jak je vidět, nakonec bylo dekódování úspěšné, jelikož cesta s váhou 2 je správná. Pokud by však vyslaná zpráva byla o jeden nebo dva bity kratší, dekódování by



Obr. 1.14: Dekódování pomocí Viterbiho algoritmu 3. krok.

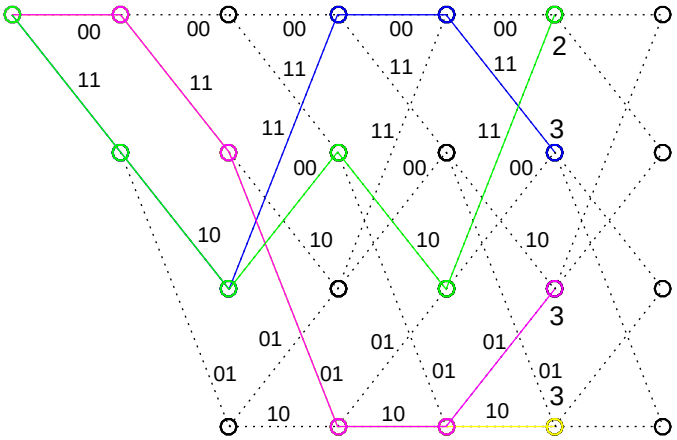


Obr. 1.15: Dekódování pomocí Viterbiho algoritmu 4. krok.

nebylo možné, jelikož v těchto případech je více cest, které mají stejnou Hammingovu váhu. Nyní už jen zpětně určíme informační bity podle cesty v mřížovém diagramu, zda se šlo horní nebo spodní cestou, $m = [1\ 0\ 1\ 0\ 0]$.

1.5 LDPC kód

Jak napovídá název samotného kódu, princip, který v roce 1963 představil *Robert Gallager* ve své dizertační práci [11], spočívá ve využití tzv. řídké paritní matice, což je matice, jež obsahuje převážně nuly. Nejdříve se pomocí geometrické konstrukce vytvoří paritní H matice, ze které se později odvodí generující matice G. Samotné kódování vychází z (1.5), kdy se informační slovo násobí s generující G maticí. Z toho plyne, že samotné kódování je již relativně jednoduché. Pro *kód s řídkou pa-*



Obr. 1.16: Dekódování pomocí Viterbiho algoritmu 5. krok.

ritní maticí – Low Density Parity Check ($LDPC$) kódy je zásadní vygenerovat G a H matici, k čemuž se využívá euklidovská nebo projektivní geometrie. Jelikož je generování matic pro $LDPC$ kódy zásadní, bude mu věnována samostatná kapitola. Další důležitou části je samotné dekódování. K tomuto účelu byla vyvinuta řada algoritmů, lišících se nejen ve složitosti, ale také účinnosti. Jednotlivé algoritmy budou podrobně představeny v následujícím textu včetně názorných příkladů.

1.5.1 Generování G a H matice

Ačkoli *Gallager* navrhl $LDPC$ kódy pro detekci a korekci chyb, neposkytl specifickou metodu pro konstrukci generujících a paritních matic algebraickou nebo systematickou metodou. Navrhl pseudonáhodnou metodu, která bude také ukázána. Později byly vynalezeny metody pro algebraickou a systematickou konstrukci založeny na konečné geometrii. Dobré $LDPC$ kódy jsou však velkých rozměrů a pro jejich výpočet se využívá počítač.

Obecně lze definovat paritní H matici jako matici s nízkým počtem jedniček jak v řádcích, tak ve sloupcích a dle [3] se dále uvádí:

- Každý řádek obsahuje ρ jedniček,
- Každý sloupec obsahuje γ jedniček,
- Počet společných jedniček mezi libovolnými dvěma sloupci není větší než jedna,
- ρ a γ jsou malé v porovnání s délkou kódu a počtem řádků v paritní H matici.

Definuje se tzv. řídkost r paritní matice H, která vyjadřuje poměr mezi počtem jedniček na řádek/sloupec ku počtu vstupních bitů v H

$$r = \frac{\rho}{n} = \frac{\gamma}{J}, \tag{1.40}$$

kde J je počet řádků v H. Matice se nazývá regulární, pokud je počet jedniček stejný jak v řádcích, tak ve sloupcích. Naopak pokud tomu tak není, matice se nazývá neregulární [9]. V této práci se budu zabývat pouze maticemi regulárními.

1.5.2 Gallagerova paritní matice

Gallager ve své práci [11] navrhl matici respektující zásady uvedené výše. Jeho matice vychází z jednotlivých submatic. Tyto submatice poté tvoří výslednou paritní matici H.

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_\gamma \end{bmatrix} \tag{1.41}$$

Označme první submatici H_1, tato má v každém řádku ρ jedniček, avšak každý sloupec obsahuje jedničku pouze jednu. Další submatice jsou vytvořeny jako permutace sloupců H_1. Z postupu této konstrukce je zřejmé, že žádné dva řádky v submatici nemají žádný společný prvek, a žádné dva sloupce nemají více než jednu společnou jedničku. Ukázka matice navržené *Gallagerem* je níže.

$$H = \left[\begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right] \tag{1.42}$$

1.5.3 Geometrická konstrukce matic

Geometrická konstrukce je založena na konečné geometrii. Také zde platí podobné zásady uvedené výše, tedy ρ značí počet jedniček na řádek a γ na sloupec, každé dva řádky mají maximálně jeden společný prvek, atd. Generovat matice geometricky lze dvěma metodami, a to pomocí:

- euklidovská geometrie,
- projektivní geometrie.

Obě metody jsou si podobné, avšak mají odlišné matematické pozadí. V práci a programu pro simulaci byla zvolena euklidovská geometrie. V následujícím textu bude teoreticky popsáno, jak matice zkonstruovat a také bude uveden názorný příklad.

EG-LDPC kódy

Vezměme v úvahu m-dimenzionální euklidovskou geometrii přes $GF(2^s)$, $EG(m, 2^s)$, jejíž podrobnější popis je uveden např. v [3]. Pro tvorbu matic LDPC kódu je zásadní použití $m = 2$, čímž vzniká 2-dimenzionální prostor. Parametr s značí stupeň kódu. EG-LDPC kódy poté mají následující parametry:

- délka kódu
$$n = 2^{2s} - 1 \tag{1.43}$$
- počet paritních bitů
$$n - k = 3^s - 1 \tag{1.44}$$
- počet informačních bitů
$$k = 2^{2s} - 3^s \tag{1.45}$$
- minimální vzdálenost
$$d_{min} = 2^s + 1 \tag{1.46}$$
- řídkost
$$r = \frac{2^s}{2^{2s} - 1} \tag{1.47}$$

Ke konstrukci matic je vhodné a u vyšších řádů nezbytné použití počítače. V tabulce 1.8 jsou zobrazeny parametry uvedené výše až do řádu $s = 7$. Jak je vidět, rozměr kódu rychle roste, a to až exponenciální řadou. Roste však také počet opravitelných chyb. V tabulce je navíc uveden počet jedniček na řádek a sloupec, jejichž počet vychází z (1.48). Cílem je vytvořit vektor v a cyklickým posuvem tohoto vektoru najít čtvercovou paritní matici H_{EG}. Vytvoření takové matice a z ní odvození matice kontrolní G_{EG} si ukážeme na příkladu.

$$\rho = \gamma = 2^s \tag{1.48}$$

Tab. 1.8: Parametry $EG - LDPC$ kódu 2-dimenzionální [3].

s	n	k	d_{min}	ρ	γ	r
2	15	7	5	4	4	0,267
3	63	37	9	8	8	0,127
4	255	175	17	16	16	0,0627
5	1023	781	33	32	32	0,0313
6	4095	3367	65	64	64	0,01563
7	16383	14197	129	128	128	0,007813

Příklad:

Pro ukázku není možné použít vyšší stupeň než $s = 2$ z důvodu velkých rozměrů. Potom mějme $m = 2$, $s = 2$ a $\mathrm{EG}(2, 2^2)$. Pole konečných prvků bude rozměrů $\mathrm{GF}(2^2)$, generované polynomem $g(x) = x^4 + x + 1$, viz tabulka 1.1.

Prvním naším cílem je najít vektor v a s jeho pomocí vytvořit kontrolní matici H_{EG}. Podle (1.48) víme, že vektor bude obsahovat celkem 4 jedničky. Pozice jedniček ve vektoru v určíme podle (1.49)

$$\{\alpha^{14} + \eta\alpha\}, \tag{1.49}$$

kde $\eta \in \mathrm{GF}(2^2)$. Prvky z konečného pole $\mathrm{GF}(4)$ se značí β a je dáno

$$\{0, 1, \beta, \beta^2\}. \tag{1.50}$$

Dosazením do následujícího vztahu získáme mocninu proměnné α, kterou dosadíme za proměnnou β a získáme tak soubor hodnot η pro dosazení do (1.49).

$$\beta = \alpha^{\frac{2^{2s}-1}{2^s-1}} = \alpha^{\frac{2^4-1}{2^2-1}} = \alpha^5 \tag{1.51}$$

Dosazením do rovnice (1.50) získáme hodnoty η pro dosazení do vztahu (1.49).

$$\{0, 1, \alpha^5, \alpha^{10}\} \tag{1.52}$$

První člen rovnice (1.49) se určí následovně a hodnoty parametru α jsou určeny podle tabulky 1.1.

$$\{\alpha^{14} + 0.\alpha\} = \{1001 + 0000\} = \{1001\} = \alpha^{14} \tag{1.53}$$

Další pozice jedniček ve vektoru v jsou $\{\alpha^7, \alpha^8, \alpha^{10}, \alpha^{14}\}$. Výsledný vektor v

bude tvaru (000000011010001) a paritní matice H_{EG} bude vypadat následovně.

$$H_{EG} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.54}$$

V dalším kroku odvodíme generující matici G_{EG}. Ta opět vznikne cyklickým posuvem vektoru g, jehož kořeny jsou α, α^2, α^3, α^4, α^6, α^8, α^9, α^{12}. Kořeny α, α^2, α^4, α^8 jsou konjugované a jejich společný minimální polynom je $p_1 = x^4+x+1$. Kořeny α^3, α^6, α^9 a α^{12} jsou také konjugované a jejich minimální polynom je $p_2 = x^4+x^3+x^2+x+1$. Potom generující vektor matice G_{EG} vznikne násobením $g = p_1 \,.\, p_2 = x^8+x^7+x^6+x^4+1$. Následně vytvoříme generující matici G_{EG} cyklickým posuvem, kdy celkový počet posuvů bude $k-1$ a na závěr ještě provedeme úpravu na systematický tvar. Matice bude poté vypadat následovně:

$$G_{EG} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{1.55}$$

1.5.4 Kódování LDPC kódu

Jak bylo uvedeno výše, kódování informačních bitů probíhá již známým způsobem, a to vynásobením informačních bitů s generující maticí dle (1.5). Ukažme si tento proces na příkladu.

Příklad:

Zakódujte informační slovo $m = [1\ 0\ 1\ 0\ 1\ 1\ 1]$ pomocí EG-LDPC kódu.

Generující G_{EG} matice byla odvozena výše, takže ji nyní použijeme ke kódování, tj. vynásobení informačního slova m s touto maticí.

$$c = m.G_{EG} = [1010111].G_{EG} = [101011110001001] \tag{1.56}$$

Kódové slovo zakódované pomocí EG-LDPC kódu je c=[1 0 1 0 1 1 1 1 0 0 0 1 0 0 1].

1.5.5 Dekódování LDPC kódu

Oproti kódování, které je u LDPC kódu relativně jednoduché, je proces dekódování o poznání složitější. Tedy pouhé ověření, zda při přenosu došlo k chybě, provedeme opět vynásobením přijatého kódového slova s kontrolní H maticí, čímž získáme hodnotu syndromu. Pokud je syndrom nulový, k chybě nejspíš nedošlo a pouze z kódového slova vyčteme informační bity. Důvod použití LDPC kódu je však v jeho schopnosti opravy chybných bitů. K tomuto účelu byla vyvinuta řada algoritmů lišících se ve složitosti, ale také účinností. Proto je většina výkonnějších algoritmů iterativních. V této práci byly implementovány celkem 3 dekódovací algoritmy, na které se nyní postupně podíváme a popíšeme si jejich funkci.

1.5.6 Dekódovací algoritmus s tvrdým rozhodováním

Tento algoritmus, jako jediný z použitých v této práci, není iterativní a v anglické literatuře je označován jako *Hard Decision*. Jeho princip spočívá v předávání bitů mezi bitovými a součtovými uzly. Fakt, že není iterativním algoritmem, je výhodou v rychlosti, tudíž je vhodný pro aplikace, které vyžadují okamžité dekódování na úkor počtu opravitelných chyb. Postup dekódování je následující:

- V prvním kroku si vypíšeme tabulku pozic podle paritní matice, na kterých je hodnota "1" a k těmto pozicím přiřadíme bit z přijatého slova ze stejné pozice.
- V kroku druhém opět nahlédneme do paritní matice a vytvoříme novou tabulku. Její řádky se budou skládat z pozic v jednotlivých sloupcích, na kterých je hodnota "1". Dále budeme pracovat s tabulkou vytvořenou v prvním kroku a počítat nové hodnoty řádků. Pro první řádek označovaný jako c_1 (první sloupec v paritní matici H_{EG}) vypočítáme hodnotu prvního sloupce jako součet modulo(2) z řádku v první tabulce na pozici odpovídající první jedničce v prvním sloupci paritní matice, vyjma aktuálně počítanou pozici.
- V posledním kroku připojíme na konec tabulky přijaté slovo a podle majoritní logiky se rozhodne o hodnotě každého bitu kódového slova.

Celý postup je relativně jednoduchý a předvedeme si ho na následujícím příkladu. Ještě je třeba uvést, že pomocí tohoto algoritmu je dle teorie možné opravit maximálně k_d chyb.

$$k_d = \frac{d_{min} - 1}{2} \tag{1.57}$$

Příklad:

Máme zakódované slovo $c = [1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1]$. Při přenosu vznikly 2 chyby na pozicích 4 a 7, takže bylo přijato slovo [1 0 1 **1** 1 1 **0** 1 0 0 0 1 0 0 1]. Pomocí algoritmu Hard Decision ověřte, zda při přenosu vznikla chyba a případně ji opravte.

Budeme postupovat podle kroků uvedených výše. Nejdříve si vytvoříme tabulku o 4 sloupcích a 15 řádcích a uložíme do ní hodnoty přijatého slova dle pozic jedniček v paritní matici. V druhém kroku budeme postupovat opět podle známého způsobu

Tab. 1.9: Vertikální krok.

vertikální krok	c_x	c_x	c_x	c_x
s_1	$c_8 \to 1$	$c_9 \to 0$	$c_{11} \to 0$	$c_{15} \to 1$
s_2	$c_1 \to 1$	$c_9 \to 0$	$c_{10} \to 0$	$c_{12} \to 1$
s_3	$c_2 \to 0$	$c_{10} \to 0$	$c_{11} \to 0$	$c_{13} \to 0$
s_4	$c_3 \to 1$	$c_{11} \to 0$	$c_{12} \to 1$	$c_{14} \to 0$
s_5	$c_4 \to 1$	$c_{12} \to 1$	$c_{13} \to 0$	$c_{15} \to 1$
s_6	$c_1 \to 1$	$c_5 \to 1$	$c_{13} \to 0$	$c_{14} \to 0$
s_7	$c_2 \to 0$	$c_6 \to 1$	$c_{14} \to 0$	$c_{15} \to 1$
s_8	$c_1 \to 1$	$c_3 \to 1$	$c_7 \to 0$	$c_{15} \to 1$
s_9	$c_1 \to 1$	$c_2 \to 0$	$c_4 \to 1$	$c_8 \to 1$
s_{10}	$c_2 \to 0$	$c_3 \to 1$	$c_5 \to 1$	$c_9 \to 0$
s_{11}	$c_3 \to 1$	$c_4 \to 1$	$c_6 \to 1$	$c_{10} \to 0$
s_{12}	$c_4 \to 1$	$c_5 \to 1$	$c_7 \to 0$	$c_{11} \to 0$
s_{13}	$c_5 \to 1$	$c_6 \to 1$	$c_8 \to 1$	$c_{12} \to 1$
s_{14}	$c_6 \to 1$	$c_7 \to 0$	$c_9 \to 0$	$c_{13} \to 0$
s_{15}	$c_7 \to 0$	$c_8 \to 1$	$c_{10} \to 0$	$c_{14} \to 0$

a vypočteme novou tabulku, ve které rovnou také přijaté slovo opravíme.

Jak je vidět, oprava byla úspěšná a bity na 4. a 7. pozici byly opraveny. Dostali jsme tedy opravené kódové slovo $d = [1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1]$. Na závěr stačí vyčíst informační bity z prvních sedmi pozic, $m = [1\ 0\ 1\ 0\ 1\ 1\ 1]$.

Tab. 1.10: Horizontální krok.

horizontální krok	s_x	s_x	s_x	s_x	rx	d
c_1	$s_2 \to 1$	$s_6 \to 1$	$s_8 \to 0$	$s_9 \to 0$	1	1
c_2	$s_3 \to 0$	$s_7 \to 0$	$s_9 \to 1$	$s_{10} \to 0$	0	0
c_3	$s_4 \to 1$	$s_8 \to 0$	$s_{10} \to 1$	$s_{11} \to 0$	1	1
c_4	$s_5 \to 0$	$s_9 \to 0$	$s_{11} \to 0$	$s_{12} \to 1$	1	**0**
c_5	$s_6 \to 1$	$s_{10} \to 1$	$s_{12} \to 1$	$s_{13} \to 1$	1	1
c_6	$s_7 \to 1$	$s_{11} \to 0$	$s_{13} \to 1$	$s_{14} \to 0$	1	1
c_7	$s_8 \to 1$	$s_{12} \to 0$	$s_{14} \to 1$	$s_{15} \to 1$	0	**1**
c_8	$s_1 \to 1$	$s_9 \to 0$	$s_{13} \to 1$	$s_{15} \to 0$	1	1
c_9	$s_1 \to 0$	$s_2 \to 0$	$s_{10} \to 0$	$s_{14} \to 1$	0	0
c_{10}	$s_2 \to 0$	$s_3 \to 0$	$s_{11} \to 1$	$s_{15} \to 1$	0	0
c_{11}	$s_1 \to 0$	$s_3 \to 0$	$s_4 \to 0$	$s_{12} \to 0$	0	0
c_{12}	$s_2 \to 1$	$s_4 \to 1$	$s_5 \to 0$	$s_{13} \to 1$	1	1
c_{13}	$s_3 \to 0$	$s_5 \to 1$	$s_6 \to 0$	$s_{14} \to 1$	0	0
c_{14}	$s_4 \to 0$	$s_6 \to 0$	$s_7 \to 0$	$s_{15} \to 1$	0	0
c_{15}	$s_1 \to 1$	$s_5 \to 0$	$s_7 \to 1$	$s_8 \to 0$	1	1

1.5.7 Dekódovací algoritmus Bit-flipping

Algoritmus Bit-flipping (BF) byl navržen *Gallagerem* [11]. Detekuje chyby při přenosu a v případě, že narazí na nesrovnalost, pokusí se ji opravit. Tento algoritmus je iterativní. Na začátku se počítá syndrom přijatého slova s pomocí paritní H_{EG} matice a pokud je syndrom nulový, bylo přijato právě vyslané slovo a není potřeba provádět opravu. Pokud je však alespoň jeden bit ve vypočteném syndromu hodnoty "1", byla detekována chyba při přenosu a algoritmus se pokusí o její opravu. Při opravě se počítá s vypočteným syndromem s, který je násoben se sloupci paritní matice a vznikne vektor S. Následně se porovnává, zda je hodnota vektoru S na pozici S_i větší, než předchozí největší hodnota. Pokud podmínka platí, pozice se uloží a na závěr se negují přijaté bity na těchto hodnotách. Znovu se vypočítá syndrom s a pokud není nulový, algoritmus se opakuje. Postup je následující:

- Vypočteme hodnotu syndromu mezi přijatým slovem a kontrolní H_{EG} maticí.
- Pokud není syndrom na všech pozicích nulový, vypočítáme pomocný vektor S podle (1.58)

$$S = s.H. \tag{1.58}$$

- Porovnáme hodnoty vektoru S a pokud hodnota na pozici S_i je větší, než předchozí největší hodnota, uložíme pozici i (hodnotám i odpovídá pro první prvek vektoru S hodnota "1", až poslednímu prvku vektoru hodnota "2^{2s}").

- Negujeme hodnoty v přijatém slově na pozicích určených v předchozím kroku.
- Vypočteme nový syndrom s a pokud není na všech pozicích nulový, opakujeme algoritmus až do maximálního zadaného počtu iterací.

Nyní si jeden cyklus uvedeného algoritmu předvedeme na příkladu.

Příklad:

Máme zakódované slovo $c = [1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1]$. Při přenosu vznikly 2 chyby na pozicích 4 a 7, takže bylo přijato slovo [1 0 1 **1** 1 1 **0** 1 0 0 0 1 0 0 1]. Pomocí algoritmu BF ověřte, zda při přenosu vznikla chyba a případně ji opravte.

Budeme postupovat podle kroků uvedených výše. Nejdříve vypočítáme s využitím (1.54) hodnotu syndromu s:

$$s = rx.H_{EG}^T = [101111010001001].H_{EG}^T = [000010011010011]. \tag{1.59}$$

Jelikož syndrom není nulový, víme, že při přenosu došlo k chybě a budeme postupovat dále k její opravě. Vypočítáme vektor S podle:

$$S = s.H_{sloupce} = [000010011010011].H_{sloupce} = [212302321201212]. \tag{1.60}$$

Z vektoru S zjistíme pozice, které budou v dalším kroku negovat bity v přijatém slově. Je to pozice 4, která je větší než předchozí hodnoty. Žádný další prvek v tomto vektoru již nemá větší hodnotu než 3, takže si uložíme pouze pozici 4. Nyní negujeme pozici 4 v přijaté zprávě a dostaneme vektor $d = [1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1]$. Dle (1.59) přepočítáme syndrom jehož hodnota je $s = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 1]$. Z toho plyne, že slovo ještě není opraveno a bude nutno provést další iteraci, v níž již dojde k opravě.

1.5.8 Dekódovací algoritmus Sum-Product

Tento algoritmus patří mezi nejúčinnější, ale jeho implementace je také nejsložitější a z hlediska výpočetního zabere nejvíce času. Hlavním cílem je najít vektor d, který bude odpovídat vyslanému slovu c, a jehož syndrom bude nulový. V této práci bude implementován zjednodušený algoritmus Sum-Product (SP), který vychází z plné verze, ale je lepší pro pochopení. Popis obou verzí lze nalézt například v [10].

Jedná se o algoritmus s měkkým rozhodováním a využívá simulaci modelu kanálu pomocí přídavného bílého Gaussovského šumu. Na začátku je potřeba udělat odhad pravděpodobností f_j^x dle přijatého symbolu podle (1.61) resp. (1.62)

$$f_j^1 = \frac{1}{1 + e^{-\frac{2Ay_j}{\sigma^2}}}, \tag{1.61}$$

potom můžeme jednoduše odvodit

$$f_j^0 = 1 - f_j^1, \tag{1.62}$$

kde y_j je hodnota přijatého bitu na pozici j. Počítá se s tím, že zpráva je modulovaná, proto je potřeba pro simulaci přijaté slovo nejdříve převést do polárního formátu s amplitudou A, která byla zvolena s hodnotou 1. Dále se počítá se standardním rozptylem σ^2, který vyjadřuje středně kvadratickou odchylku šumu s normálním rozdělením v přenosovém kanále. V práci a programu se počítá s hodnotou $\sigma^2 = 0{,}8$ podle [10].

Pro výpočet koeficientů R_{ij}^x je nutné stanovit funkce Q_{ij}^x. Bylo však dokázáno [12], že ve zjednodušené verzi algoritmu můžeme určit tyto funkce jako

$$Q_{ij}^0 = f_j^0, \tag{1.63}$$

$$Q_{ij}^1 = f_j^1. \tag{1.64}$$

Tato verze provádí výpočet iterativním provedením kroků, a to horizontálním a vertikálním způsobem, kdy se jednotlivé koeficienty určují podle paritní H matice. Hodnota δQ_{ij} se určí podle

$$\delta Q_{ij} = Q_{ij}^0 - Q_{ij}^1, \tag{1.65}$$

a hodnota δR_{ij} je dána rovnicí (1.66)

$$\delta R_{ij} = \prod_{j' \in N(i) \setminus j} \delta Q_{ij'}. \tag{1.66}$$

V dalším kroku se určí koeficienty R_{ij}^0 a R_{ij}^1, které jsou nutné pro odhad přijatých bitů.

$$R_{ij}^0 = 1/2.(1 + \delta R_{ij}) \tag{1.67}$$

$$R_{ij}^1 = 1/2.(1 - \delta R_{ij}) \tag{1.68}$$

Koeficienty Q_{ij}^0 a Q_{ij}^1 jsou aktualizovány ve vertikálním kroku. Jsou určeny pro každý pár uzlu i, j a pro všechny možné hodnoty proměnné x, které jsou v binárním případě $x = 0$ nebo $x = 1$. Vypočteme

$$Q_j^x = \alpha_j f_j^x \prod_{i \in M(j)} R_{ij}^x, \tag{1.69}$$

kde $M(j)$ odkazuje na bitové uzly, koeficient f_j^x značí pravděpodobnosti určené výše a α_j se vypočítá dle rovnice

$$\alpha_j = \frac{1}{f_j^0 \prod_{i \in M(j)} R_{ij}^0 + f_j^1 \prod_{i \in M(j)} R_{ij}^1}. \tag{1.70}$$

Výsledná hodnota vektoru d je určena podle (1.71), tzn. vyslaný bit je určen podle velikosti Q_j^x.

$$d_j = max(Q_j^x) \tag{1.71}$$

Pro další iteraci je nutné aktualizovat koeficienty $f_j^0 = Q_j^0$ a $f_j^1 = Q_j^1$. Další iterace se bude počítat pouze v případě, že nově vypočítaný syndrom nebude nulový a zároveň nebyl překročen maximální nastavený počet iterací. Pro názornost si postup ještě předvedeme na příkladu.

Příklad:

Máme zakódované slovo $c = [1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 1]$. Při přenosu vznikly 2 chyby na pozicích 4 a 7, takže bylo přijato slovo [1 0 1 **1** 1 1 **0** 1 0 0 0 1 0 0 1]. Pomocí algoritmu SP ověřte, zda při přenosu vznikla chyba a případně ji opravte.

Přijaté slovo je modulováno pomocí dvoustavové BPSK modulace s amplitudou ± 1 a dále simulován průchod AWGN kanálem s rozptylem $\sigma = 0{,}8$ a střední hodnotou $\mu = 0$. Poté se vypočte dvojice pravděpodobností f_j^0 (1.62) a f_j^1 (1.61), které jsou zobrazeny v tabulce 1.11.

Tab. 1.11: Hodnoty proměnných f_j^0 a f_j^1.

pozice j	proměnná f_j^0	proměnná f_j^1
1	0,0399	0,9601
2	0,9814	0,0185
3	0,0236	0,9764
4	0,0589	0,9410
5	0,0495	0,9504
6	0,0549	0,9451
7	0,9653	0,0347
8	0,0321	0,9680
9	0,9686	0,0314
10	0,9653	0,0347
11	0,9580	0,0419
12	0,0473	0,9526
13	0,9046	0,0954
14	0,9495	0,0505
15	0,0260	0,9740

Definují se koeficienty Q_{ij}^0 a Q_{ij}^1 jako

$$Q_{ij}^0 = f_j^0 a\ Q_{ij}^1 = f_j^1.$$

Následuje horizontální krok, ve kterém se podle postupu uvedeného výše určí koeficienty R_{ij}^0 a R_{ij}^1.

Na závěr provedeme vertikální krok, ve kterém dojde k novému určení koeficientů Q_j^0 a Q_j^1, podle kterých se určuje vektor d. Pro jejich výpočet je však nutný parametr

α_j, jehož hodnota je stejná pro všechny řádky v paritní matici, a proto je v následující tabulce uvedena jeho hodnota v jednom sloupci.

Tab. 1.12: Hodnoty proměnných α_j, Q_j^0, Q_j^1 a d.

pozice j	proměnná α_j	proměnná Q_j^0	proměnná Q_j^1	d
1	73,99	0,0933	0,9067	1
2	12,11	0,9996	0,0004	0
3	71,82	0,0220	0,9780	1
4	78,47	0,8057	0,1943	0
5	1,89	0,00001	0,9999	1
6	49,34	0,0352	0,9648	1
7	123,09	0,2782	0,7218	1
8	44,25	0,0341	0,9659	1
9	14,59	0,9998	0,0002	0
10	79,22	0,9595	0,0405	0
11	2,08	0,9999	0,00001	0
12	12,82	0,0005	0,9995	1
13	72,30	0,9200	0,0799	0
14	8,11	0,9988	0,0012	0
15	72,42	0,0170	0,9830	1

Jak je vidět v tabulce 1.12, podle majoritní logiky bylo rozhodnuto o přijatém vektoru d, a jelikož nový syndrom je nulový, bylo přijaté slovo opraveno správně již v prvním kroku tohoto algoritmu.

2 Praktická část - popis programu

Jak bylo popsáno v úvodu, cílem práce je vytvořit program pro demonstraci kanálového kódování. V první kapitole byla podrobně rozepsána potřebná teorie vybraných kanálových kódů. Program má sloužit jako učební pomůcka, proto byl kladen důraz na názornost, aby si uživatel mohl představit, co se s informačními bity před přenosem děje, tedy jak se kódují. Graficky je znázorněn také průchod kódového slova přenosovým kanálem. Následuje hlavní část, a tou je samotné dekódování, které dá uživateli možnost nahlédnout, jak prakticky daný kód funguje a kolik chyb je schopen opravit. V této kapitole bude popsáno, jak byl program napsán, ukázáno uživatelské rozhraní a rozepsáno, jak program ovládat.

2.1 Vlastní řešení

Samotný kód, kódování informace, průchod přenosovým kanálem a zpětné dekódování je obvykle složeno z několika dílčích kroků, které je třeba provést. K tomu byly napsány dílčí funkce, jejichž vhodným složením je možné demonstrovat princip jednotlivých kódů. Postup simulace kanálového kódování je stejný jako v případě reálného přenosového řetězce. Uživatel si nejprve zvolí kód, který bude používat, poté, pokud je to potřeba, definuje některé náležitosti daného kódu, jako může být například počet paritních bitů, rychlost kódu nebo generující polynom. V dalším kroku zadá samotné informační bity, které chce přenést kanálem a pro tuto potřebu kódovat. V reálném systému tomu samozřejmě předchází zdrojové kódování, které není předmětem této práce. Následuje aplikace kódovacího algoritmu a vypsání kódového slova. V tomto okamžiku je na uživateli, zda provede rovnou dekódování, nebo vnese do kódového slova chybu, což se běžně při průchodu přenosovým kanálem stane. To je možno udělat buď ručně a editovat zakódované slovo na libovolném místě, nebo stiskem tlačítka aplikovat na slovo AWGN šum a sledovat, co se s ním stane. Přístup vkládání aditivního bílého Gaussovského šumu je stejný u všech kódů, kterým se věnuje tato práce. Případně můžeme vnést chyb více a pozorovat, kolik jich zvládne algoritmus opravit, kdy už je oprava chybná, či ji nelze vůbec provést. Po spuštění dekódování je oznámeno, zda došlo při přenosu k chybě a pakliže ano, jestli ji bylo možné opravit. Někdy se stane, pokud je chyb příliš, že dojde k opravě na jiné kódové slovo, jež by bylo vyhodnoceno jako vyslané. Na všechny tyto možnosti program upozorní. Pokud bylo dekódování úspěšné, jsou na závěr vypsány dekódované informační bity.

2.2 Hlavní okno

Pro aktivaci programů jednotlivých kódů slouží hlavní okno, jehož náhled je na obrázku 2.1. Hlavní okno se zapne spuštěním souboru s názvem `"Program_Pro_Demonstraci_Kanaloveho_Kodovani"`.m v programu Matlab. Zde uživatel volí pomocí čtyř tlačítek požadovaný kód. Popis programů pro jednotlivé kódy je předmětem následujících kapitol.

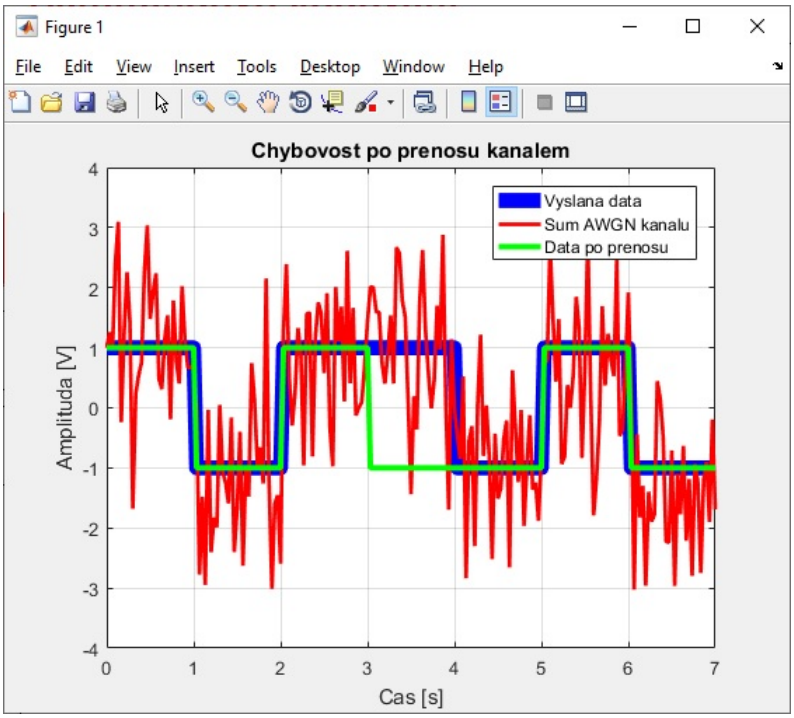
Obr. 2.1: Náhled na hlavní okno programu.

2.3 Program pro Hammingův kód

Vychází se z teorie popsané v kapitole 1.2, která se věnuje právě Hammingovu kódu. Princip kódování je založen na násobení s generující G maticí. Náhled na hlavní okno programu je na obrázku 2.4. V prvním kroku je potřeba zadat počet paritních bitů, který musí být větší nebo roven číslu 3. Následně program uživatele vyzve, aby doplnil daný počet informačních bitů. To je možné buď ručně, nebo stiskem tlačítka "GENERUJ INF. BITY". Pokud je zadán správný počet míst, dojde po kliknutí na tlačítko "KÓDUJ" k zobrazení generující G matice. V případě, že je zadán počet paritních bitů větší než 4, velikost matic rychle roste do rozsáhlých a nepřehledných rozměrů. Pro názornou ukázku a pochopení však počet paritních bitů 3 nebo 4 naprosto dostačuje. Proto, je-li zadán větší rozměr kódu, je program stále možné využít ke kódování a simulaci, ale zobrazí se hláška, že jsou matice příliš velké a pro jejich zobrazení je nutné zadat počet paritních bitů z intervalu <3,4>. Pokud je vše v pořádku, stisknutí tlačítka "KÓDUJ" způsobí vynásobení zadaného informačního slova s generující G maticí, což je také zobrazeno v prostoru pro to určeném a výsledek se ukáže v políčku "ZAKÓDOVANÉ SLOVO".

V další části je možné simulovat průchod přenosovým kanálem. To je možné dvěma způsoby, a sice manuálně editovat zakódované slovo, kdy můžeme změnit bit

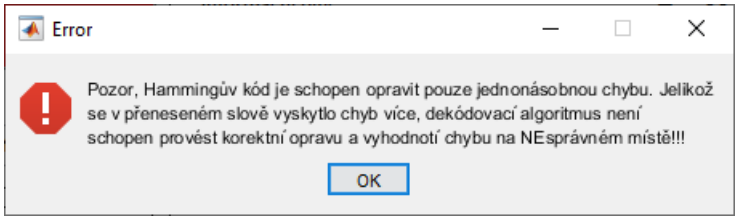
na libovolném místě a vytvořit tak chybu, případně chyb několik. Druhou možností je zadat hodnotu odstupu signálu od šumu v decibelech a stiskem tlačítka "ŠUM" dojde k aplikaci aditivního bílého Gaussovského šumu na zakódované slovo. Počet vzniklých chyb je vypsán v políčku vedle zakódovaného slova a slovo přenesené kanálem je zapsáno v poli "PŘENOS KANÁLEM". Navíc je zobrazen graf, k nahlédnutí na obrázku 2.2, který dokresluje působení šumu na zakódované bity. Modrou barvou je znázorněno vyslané kódové slovo, červeně šum, který na průchozí informaci působí, a zeleně jsou zobrazena přijatá data.



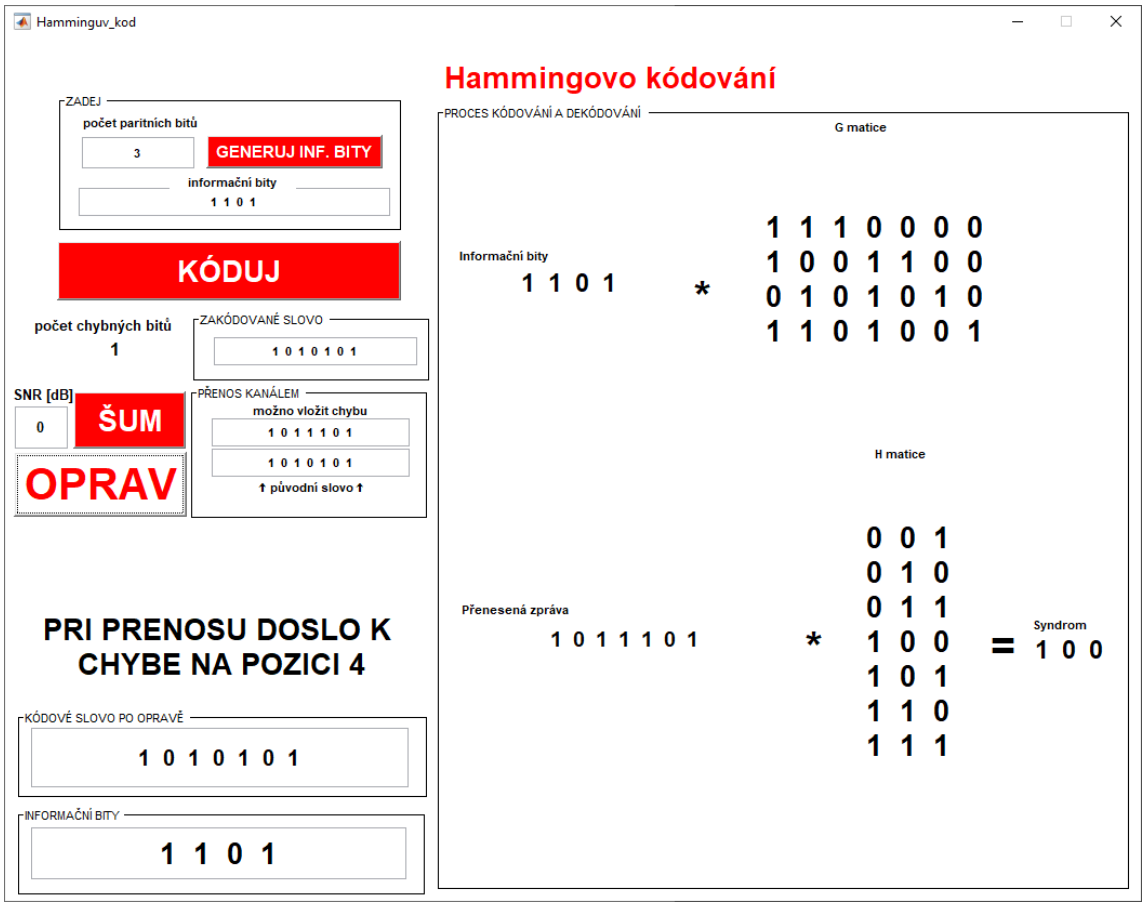
Obr. 2.2: Graf simulace přenosového AWGN kanálu.

V posledním kroku je k dispozici tlačítko "OPRAV". Po jeho stisknutí dojde k vynásobení přijatého slova s kontrolní H maticí a je vypočítán syndrom, který určuje pozici chyby v přijatém slově. Násobení kódového slova s paritní maticí je opět znázorněno v pravé části programu. Vyjde-li syndrom nulový, bylo s vysokou pravděpodobností přijato vyslané kódové slovo. V opačném případě je syndromem indikovaná pozice chyby, jež je opravena. Může nastat situace, že při přenosu vznikne chyb více a potom není oprava správná, jelikož Hammingův kód je schopen opravit pouze jednu chybu v přijatém slově. V takovém případě je vyhodnoceno, že došlo k jedné chybě na nesprávné pozici, nebo pokud je chyb více, může být dokonce přenesené slovo zasaženo chybou prohlášeno za vyslané, neboť hodnota syndromu vyjde nulová. To je daň za jednoduchost Hammingova kódu. Všechny tyto možnosti

jsou v programu ošetřeny a uživatel může buď záměrně zadávat různý počet chyb a sledovat, co se bude dít, kolik chyb je možno opravit a jak na to algoritmus reaguje, nebo v případě nepozornosti je obsluha na danou skutečnost upozorněna. Informace o průběhu přenosu je vypsána pod dekódovacím tlačítkem. Pokud dojde k nějaké nestandardní situaci, je zobrazeno upozorňující vyskakovací okno, viz obrázek 2.3. Na závěr se znázorní opravené kódové slovo, ze kterého jsou určeny informační bity. Na obrázku 2.4 je náhled okna programu po dokončení dekódování.



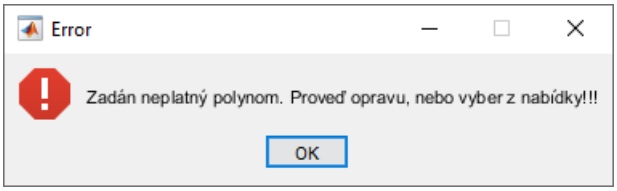
Obr. 2.3: Vyskakovací okno upozorňující na chybu při opravě.



Obr. 2.4: Okno programu Hammingova kódu.

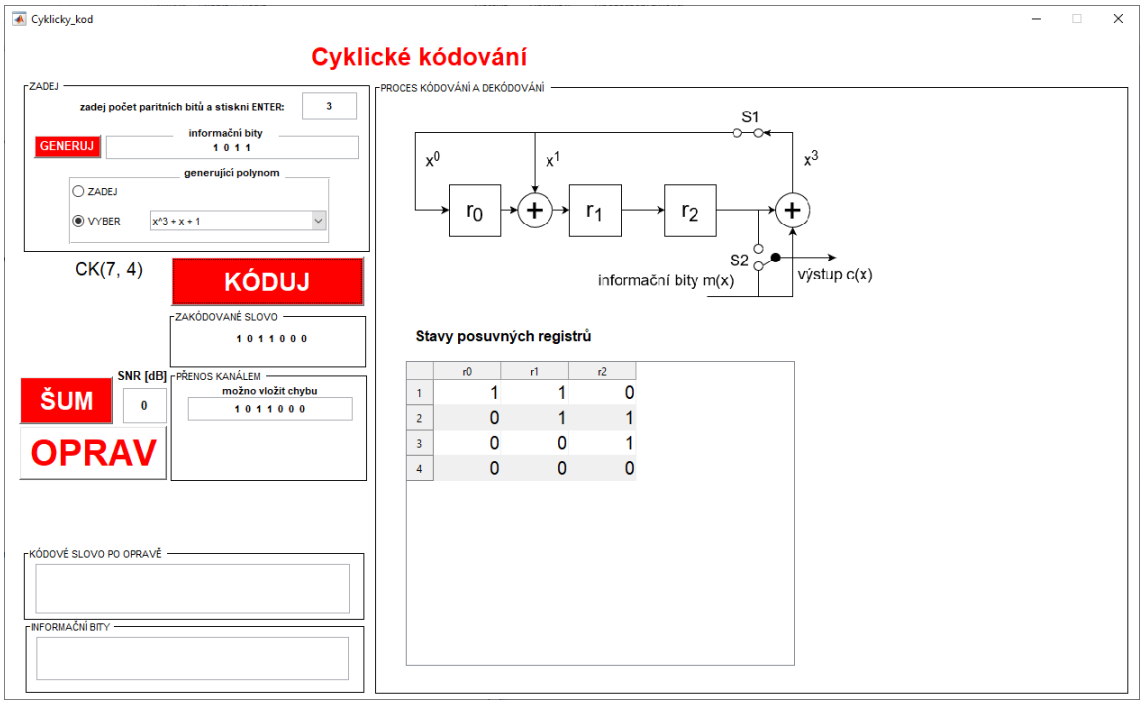
2.4 Program pro Cyklický kód

Základní poznatky k cyklickému kódu jsou popsány v kapitole 1.3. Je několik možností, jak kódovat a dekódovat informace pomocí tohoto kódu. Pro názornou ukázku bude nejpraktičtější ukázat postup pomocí kodéru a dekodéru složených z posuvných registrů. Po spuštění programu je třeba zadat počet zabezpečujících bitů, ze kterých program vyhodnotí, kolik je potřeba zadat bitů informačních. Ty je možné zadat buď opět ručně, nebo stiskem tlačítka "GENERUJ" nechat vypsat náhodnou posloupnost odpovídající délky. Navíc je nutno určit generující polynom $g(x)$. Zde jsou uživateli nabídnuty dvě možnosti. Buď může zadat polynom ručně, kdy se zadává formou binární logiky, takže vložení "1" značí, že prvek odpovídající dané mocnině je obsažen a "0" nikoli. Po zadání polynomu je nejvyšší mocnina vlevo a prvek s mocninou nula vpravo. Například polynom $x^3 + x^2 + 1$ se vloží jako posloupnost bitů oddělených mezerou [1 1 0 1]. Program navíc hlídá, aby zadaný polynom odpovídal požadavkům na generování cyklického kódu. Pokud polynom není správný, zobrazí se upozorňující hláška, která vyzve uživatele k opravě, nebo výběru z nabídky, viz obrázek 2.5.



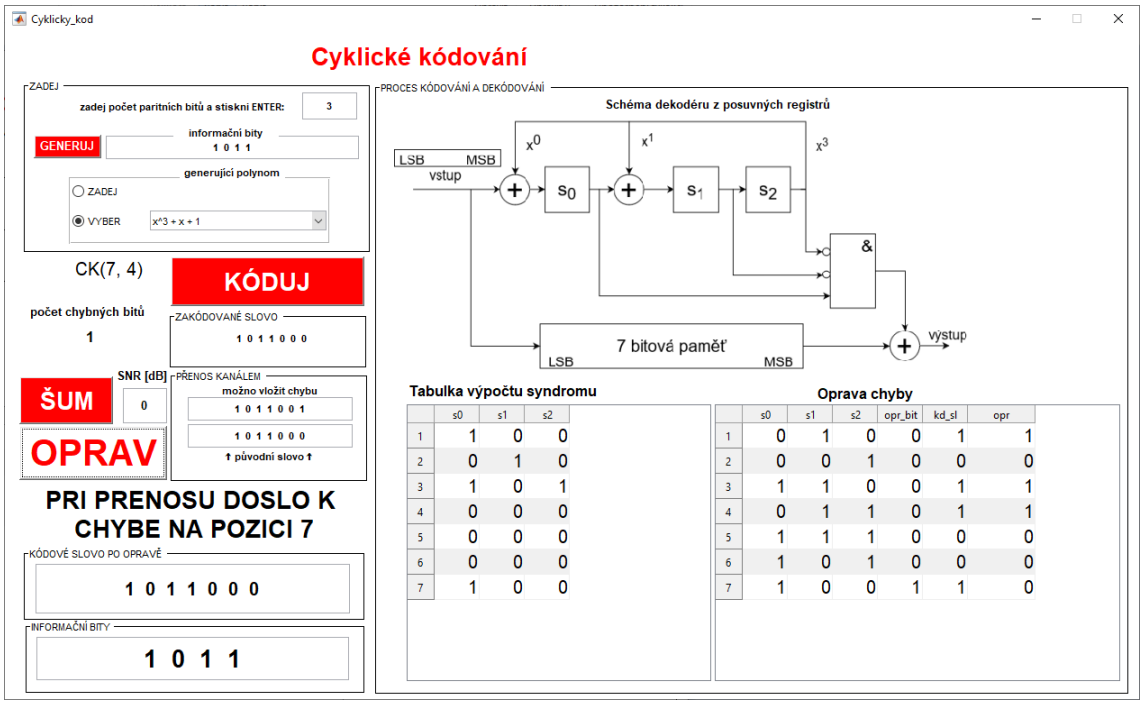
Obr. 2.5: Hláška při zadání nesprávného polynomu.

Výběr polynomu z předvolené nabídky je druhou možností volby generujícího polynomu. Jsou zobrazeny vždy jen polynomy odpovídající počtu zadaných paritních bitů. Tím jsou dokončeny kroky nutné pro kódování. Stiskem tlačítka "KÓDUJ" dojde k zobrazení blokového kodéru složeného z posuvných registrů podle zadaného polynomu a tabulky, která zachycuje stavy jednotlivých registrů během kódování. V políčku "ZAKÓDOVANÉ SLOVO" se zobrazí výsledek kódování, který odpovídá na nejvyšších pozicích zadaným informačním bitům a k nim se připojí zadaný počet paritních bitů, jejichž hodnoty odpovídají stavům posuvných registrů po nasunutí všech informačních bitů. Stejný výsledek se zapíše do pole "PŘENOS KANÁLEM". Zakódované slovo je možné editovat a simulovat tak průchod přenosovým kanálem, stejně jako v případě Hammingova kódování. Nebo zadáním SNR a stiskem tlačítka "ŠUM" aplikovat na kódové slovo aditivním bílý Gaussovským šum. Tento přístup je stejný u všech kódů, kterým se věnuje tato práce. Náhled okna programu v režimu kódování je zobrazen na obrázku 2.6.



Obr. 2.6: Okno programu cyklického kódu pro kódování.

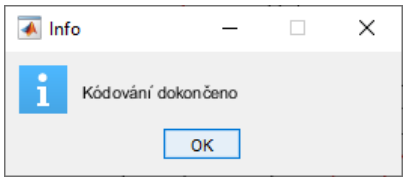
Stiskem tlačítka "OPRAV" zahájíme proces dekódování přijatého slova. Opět se zobrazí ekvivalentní obvod složený z posuvných registrů, binárních sčítaček a paměti, který zajistí dekódování. V případě, že je přijato vyslané kódové slovo, je hodnota registrů po průchodu celého přijatého slova nulová. Není tedy potřeba provádět proces opravy a rovnou se vyčtou informační bity. Případ, kdy je v přijatém slově chyba, indikují nenulové stavy registrů po nasunutí celého kódového slova do obvodu dekodéru. Oprava probíhá způsobem, že se na vstup dekodéru přivádí bity s nulovou hodnotou a stavy registrů se po průchodu logickým obvodem sčítají s přijatým kódovým slovem, které je uloženo v paměti. Počítá se opravný bit, který má hodnotu "1" pouze v případě, že je na této pozici zaznamenána chyba. Oprava se provede sečtením modulo(2) vektoru s opravným bitem a přijatého kódového slova. Také cyklický kód, stejně jako Hammingův, je schopen opravit pouze jednonásobnou chybu. Proto, pokud je ve slově chyb více, dojde k vyhodnocení chyby na nesprávné pozici. Při velkém počtu chyb může být přijaté slovo vyhodnoceno dokonce jako bezchybné. Všechny tyto možnosti jsou v programu ošetřeny a uživatel si může vyzkoušet, jak dekódování probíhá a kolik chyb je možné opravit. Na závěr je vypsáno opravené kódové slovo a vyčteny informační bity. Na obrázku 2.7 je zaznamenán náhled okna programu při dekódování.



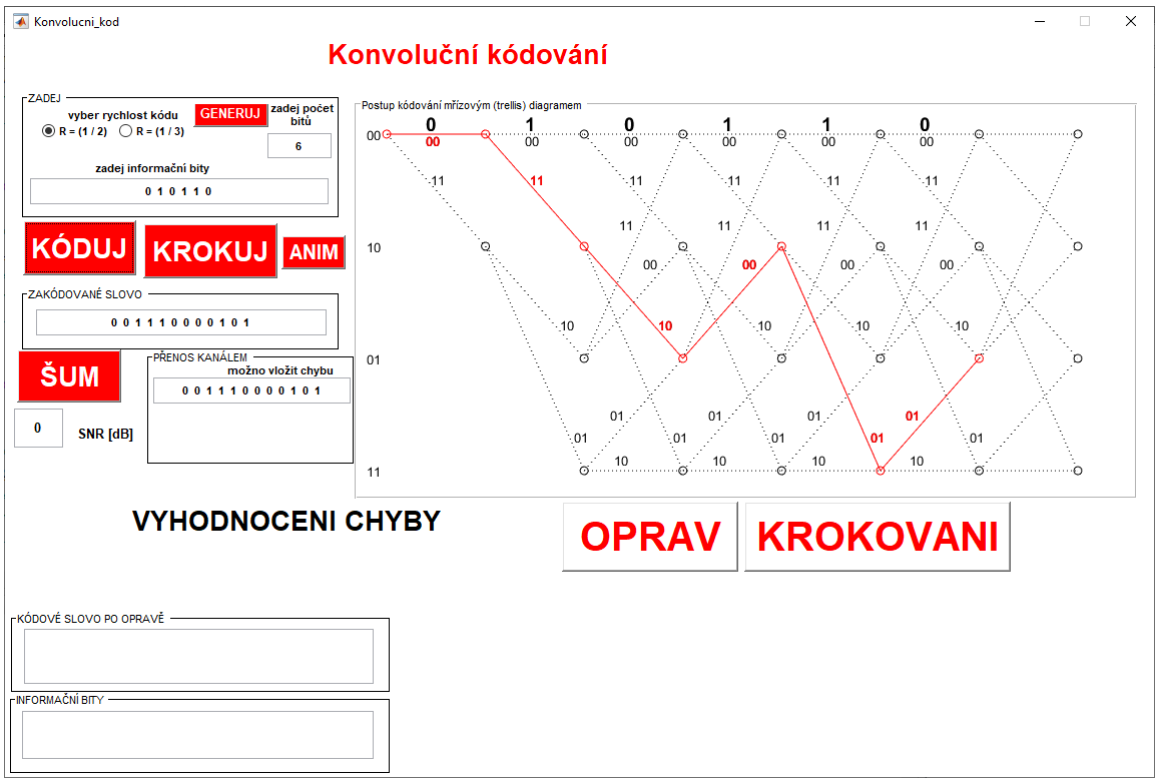
Obr. 2.7: Okno programu cyklického kódu pro dekódování.

2.5 Program pro Konvoluční kód

Princip fungování konvolučního kódu je popsán v kapitole 1.4. Opět existují různé způsoby, jak provádět kódování a dekódování. Pro názornou ukázku v programu bylo zvoleno kódování pomocí mřížového diagramu. V úvodu je třeba zvolit rychlost kódu, kdy je nabízena možnost $R = 1/2$ nebo $R = 1/3$. Dále se zadá pouze libovolný počet informačních bitů, které mají být kódovány. To je opět možné provést buď ručně, nebo stiskem tlačítka "GENERUJ". Vše potřebné pro proces kódování máme splněno a může se přistoupit k procesu kódování. Zde jsou k dispozici tři možnosti, kdy každé odpovídá jedno samostatné tlačítko. Stisk tlačítka "KÓDUJ" provede zakódování informačních bitů do kódového slova, které je vypsáno v příslušném poli. Navíc se zobrazí mřížový diagram, ve kterém je vyznačen postup kódování s červeně zvýrazněnou cestou odpovídající informačním bitům. Náhled okna programu v režimu kódování je zobrazen na obrázku 2.9. Druhou možností je využít tlačítko "KROKUJ", které při prvním stisku zobrazí prázdný mřížový diagram s vypsanými informačními bity nad ním. Zde je možnost pro uživatele promyslet si každý krok kódování a ověřit si stiskem tlačítka svou úvahu. S každým stiskem se červeně zobrazí nový krok v mřížovém diagramu. Jakmile jsou zakódovány všechny informační bity, objeví se hláška indikující tuto událost viz obrázek 2.8. Poslední možností je použít tlačítko "ANIM", které provede kódování po krocích v půl sekundových intervalech.



Obr. 2.8: Vyskakovací okno programu o dokončení kódování.

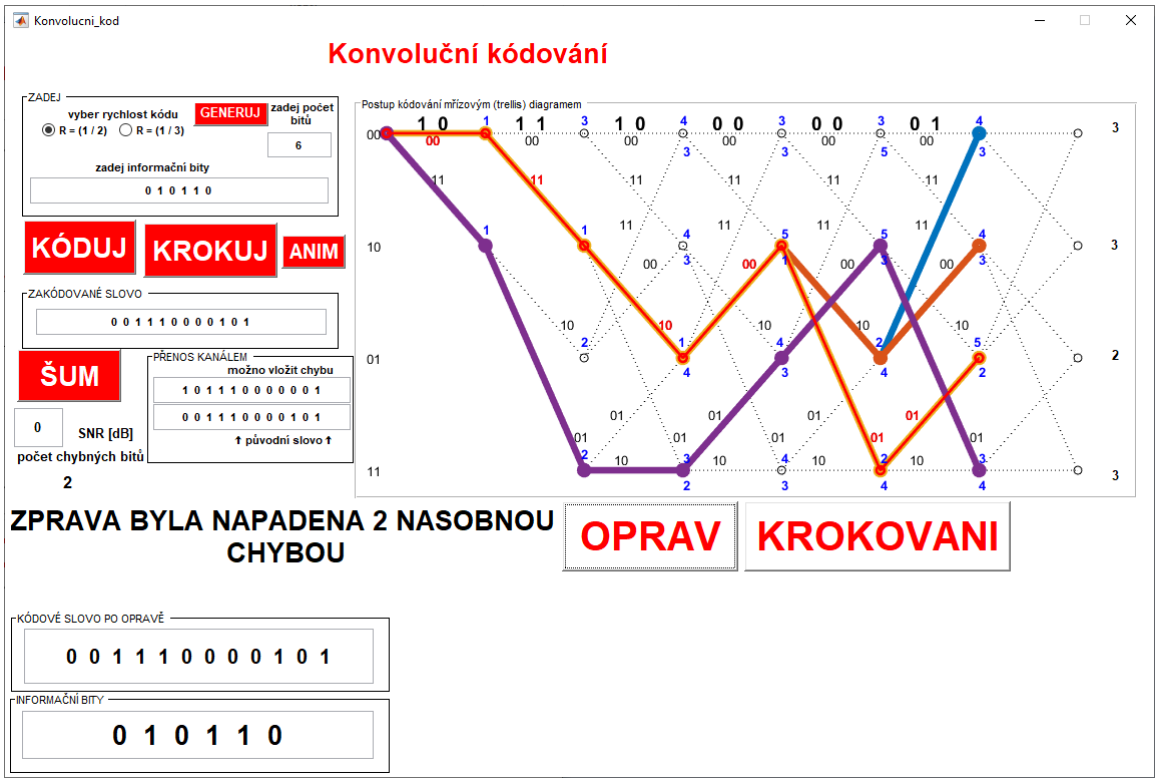


Obr. 2.9: Okno programu konvolučního kódu v režimu kódování.

Simulaci zanesení chyby vlivem průchodu přenosovým kanálem, obdobně jako v předchozím případě, je možné provést opět buď ručně nebo stiskem tlačítka "ŠUM".

Následným stiskem tlačítka "OPRAV" dojde k samotnému dekódování. Dekódování je založeno na Viterbiho algoritmu, který prochází všechny možné cesty a počítá Hammingovu vzdálenost od přijatého slova. V poli pro vykreslení se zobrazí opět mřížový diagram s nejpravděpodobnějšími cestami, kterými mohly být informační bity kódovány. Ke každému uzlu je vypsána průběžná Hammingova váha vůči přijatému slovu. Pro další krok se vybere menší ze dvou možných a kumulativně se vypočte nová. Takto se dojde až na závěr, kdy jsou vyhodnoceny čtyři možné cesty s nejmenší celkovou Hammingovou váhou. Jako původní kódové slovo se určí cesta, jejíž Hammingova váha je nejmenší. Pokud je váha některé cesty nulová, je velmi

pravděpodobné, že bylo přijato právě vyslané kódové slovo. Na druhou stranu v případě, pokud je stejná hodnota pro několik cest, nelze s jistotou určit vyslané kódové slovo. Také může dojít k případu, že bude slovo obsahovat velký počet chyb a některá z cest bude mít menší celkovou Hammingovu váhu než cesta vyslaného slova. V tomto případě je slovo opraveno špatně. Na všechny tyto případy program upozorní, proto si může uživatel libovolně zkoušet, kolik chyb je možné opravit a jak na ně algoritmus reaguje. Program je sestaven tak, že uživatele celým procesem provede, takže bude vždy vědět, k jakému případu právě došlo. Na obrázku 2.10 je zaznamenán náhled okna programu v režimu dekódování.

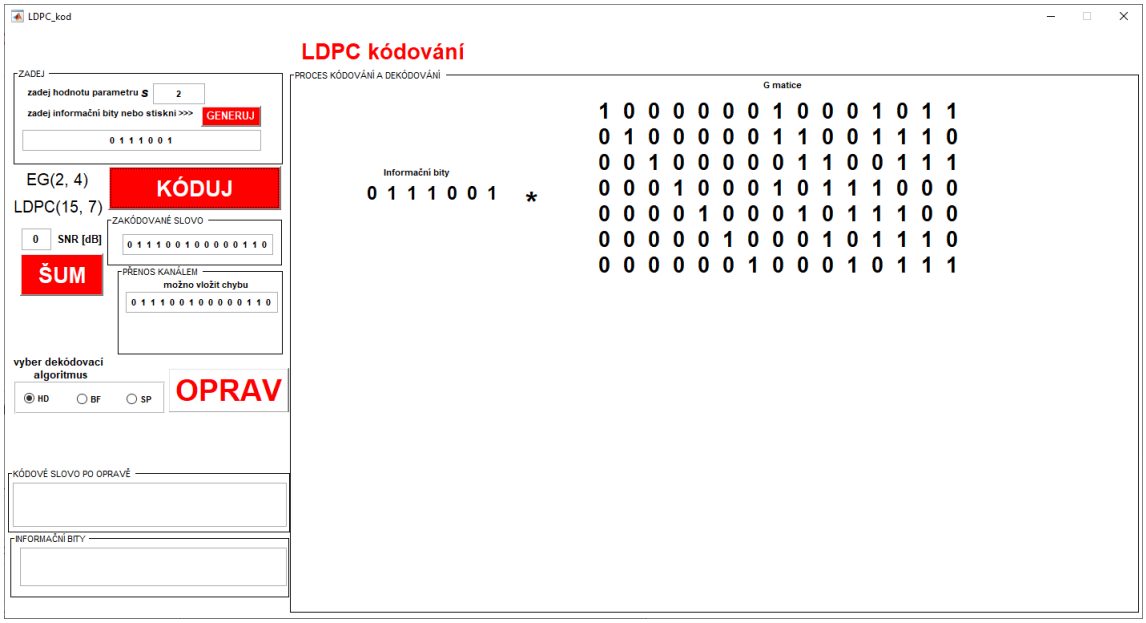


Obr. 2.10: Okno programu konvolučního kódu v režimu dekódování.

2.6 Program pro LDPC kód

LDPC kódy patří k nejefektivnějším, ale také nejnáročnějším pro samotnou realizaci. Vytváření generující a paritní matice jsou věnovány samostatné kapitoly 1.5.1, 1.5.2 a 1.5.3. Další kapitoly se zabývají kódováním a především dekódováním. Samotný program je koncipován v podobném stylu jako všechny předchozí. Výchozí okno po procesu kódování je na obrázku 2.11. V úvodu je vyžadováno zadání parametru s, což je řád kódu. Pro interaktivní zobrazení je možné pouze $s = 2$, jelikož rozměry

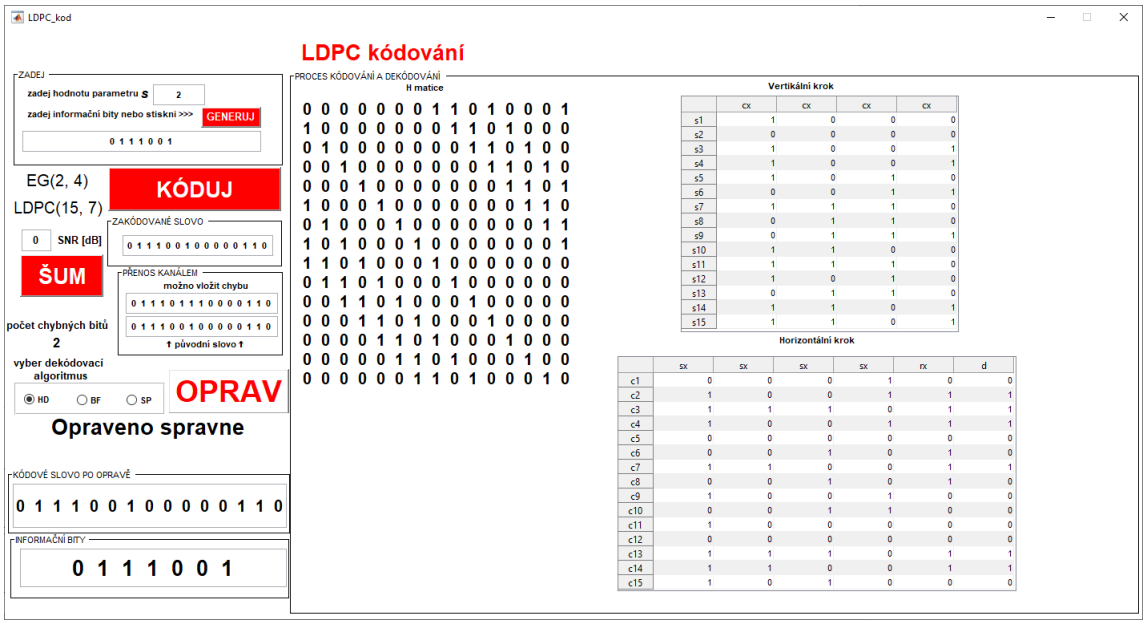
matic rychle rostou, například pro $s = 3$ má již paritní matice 63 řádků, a zobrazení by nebylo reálné. Další podrobnosti, jak rostou rozměry kódu, viz tabulka 1.8. Fungování však není řádem omezeno, pouze nedojde k zobrazení průběhu kódování/dekódování. Dále je třeba zadat pouze informační bity. Program stejně jako u všech předchozích kódů hlídá, aby byly v poli pro informační bity zadány pouze binární data oddělena mezerou, jinak dojde k upozornění pomocí vyskakovacího okna. Stisk tlačítka "KÓDUJ" pro případ $s = 2$ zobrazí generující G matici násobenou informačními bity. Výsledek kódování je zapsán do pole "ZAKÓDOVANÉ SLOVO".



Obr. 2.11: Okno programu LDPC kódu v režimu kódování.

Následuje simulace průchodu kódového slova přenosovým kanálem, kdy již klasicky můžeme slovo editovat ručně nebo pomocí tlačítka "ŠUM" simulovat působení bílého Gaussovského šumu.

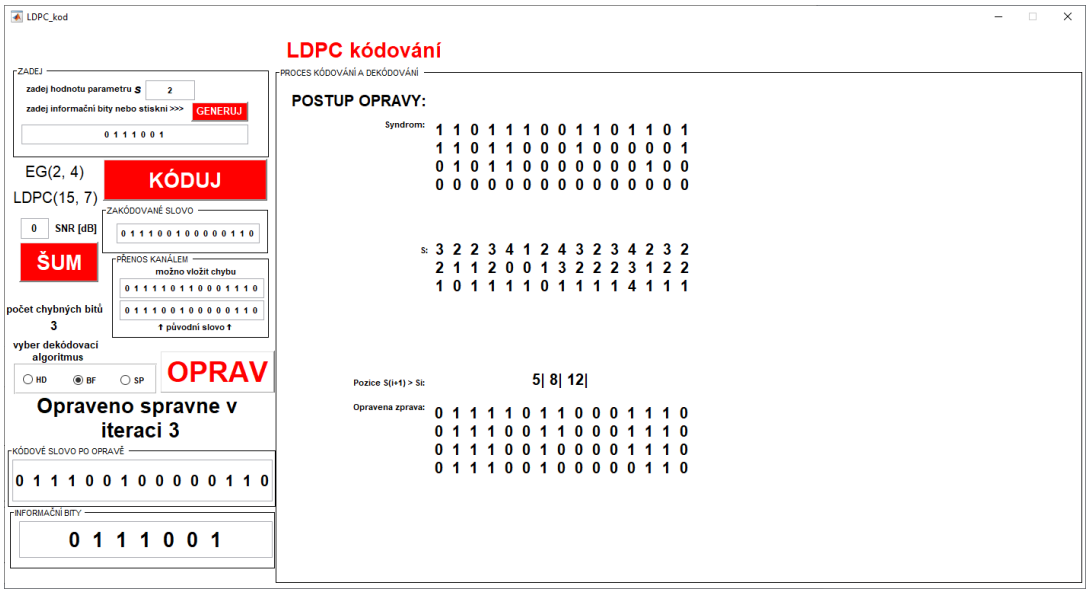
Pro dekódování byly zvoleny celkem 3 algoritmy, jejichž teorie i příklady jsou uvedeny výše. Před dekódováním se vybere požadovaný dekódovací algoritmus a stiskem tlačítka "OPRAV" se spustí proces opravy. Pokud se zatrhne možnost HD, značící algoritmus Hard Decision, probíhá dekódování následovně. Na začátku se opět zkontroluje, zda jsou v poli pro přenesené slovo zadány pouze binární hodnoty. Pokud ano, spočítá se syndrom a vyhodnotí se, zda při přenosu došlo k chybě. Jestliže je detekována chyba, zobrazí se paritní H matice a tabulky s horizontálním a vertikálním krokem, ve kterých si může uživatel číst a zorientovat se v procesu opravy. Pokud je chyb více, než odpovídá teorii podle (1.57), je pomocí syndromu vyhodnoceno, že slovo není možné opravit. Náhled okna s využitím HD algoritmu je na obrázku 2.12.



Obr. 2.12: Okno programu LDPC kódu v režimu dekódování s použitím HD algoritmu.

V případě volby dekódovacího algoritmu BF se jedná o iterativní algoritmus. V pravé části programu, kde je vyhrazeno místo pro zobrazení procesu kódování a dekódování, je ukázán proces opravy. Postupuje se dle pravidel uvedených výše a v případě detekce chyby je zobrazen vypočtený syndrom, který se dále použije k výpočtu vektoru S, jenž je rovněž vysázen na obrazovku. Pod tímto vektorem jsou vypsány pozice, jenž jsme odvodili z vektoru S, které se v každé iteraci negují, odděleny svislou čarou. V poslední části je postupně vypsána přijatá zpráva a na každém dalším řádku se již negují přijaté bity podle vyhodnocených pozic. Tímto postupem je zobrazeno opravování chybou zasaženého přijatého slova. Dekódování končí v případě, že je z opraveného slova vypočítán nulový syndrom, nebo dosaženo maximálního počtu iterací. Opět platí, že v případě velkého počtu chyb, které již není algoritmus schopen opravit, může dojít k opravě na jiné kódové slovo, jehož syndrom bude samozřejmě nulový a oprava by byla vyhodnocena jako úspěšná. Jak už je zvykem, program na tuto skutečnost upozorní. Náhled okna s využitím BF algoritmu najdeme na obrázku 2.13.

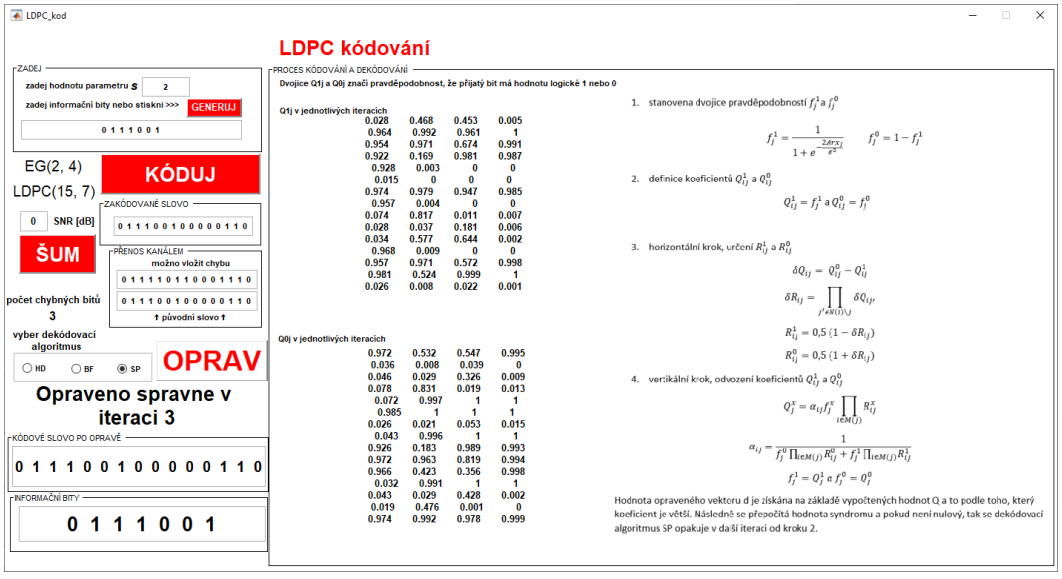
Posledním možným dekódovacím algoritmem je zjednodušený SP. Tento je opět iterativní a skládá se z několika kroků vedoucích k opravě kódového slova. Výpis všech bodů postupu by byl nepřehledný a nic neříkající, proto bylo zvoleno vypisovat hodnoty pravděpodobnosti Q přijatých bitů, ze kterých se určuje opravený vektor d podle toho, která pravděpodobnost má větší hodnotu. Navíc je podrobně vypsán postup celého dekódování pro dokreslení komplexnosti celého algoritmu. Náhled okna



Obr. 2.13: Okno programu LDPC kódu v režimu dekódování s použitím BF algoritmu.

s využitím SP algoritmu je na obrázku 2.14. Opět platí, že program upozorní, pokud není algoritmus schopen vzniklé chyby opravit, případně dojde k opravě na jiné kódové slovo.

Výhodou realizace více dekódovacích algoritmů je možnost vzájemného porovnání.

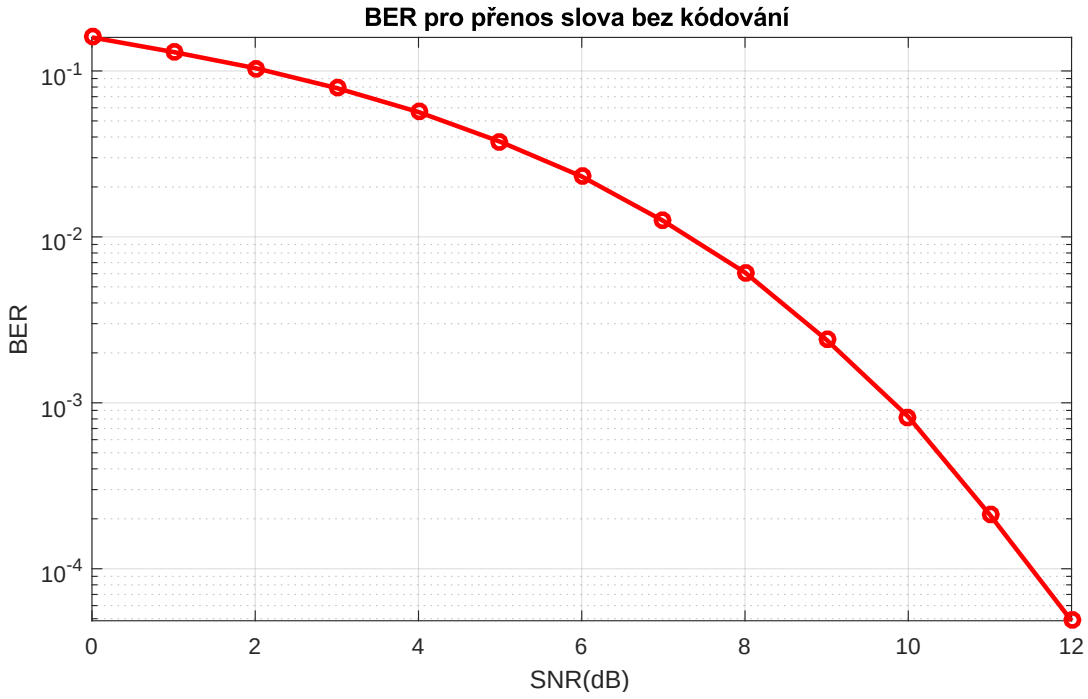


Obr. 2.14: Okno programu LDPC kódu v režimu dekódování s použitím SP algoritmu.

3 Porovnání a zhodnocení chybovosti

Tato kapitola se věnuje vzájemnému porovnání jednotlivých kódovacích a dekódovacích algoritmů, především byl kladen důraz na vyhodnocení bitové chybovosti BER. Ta vyjadřuje poměr mezi chybně přijatými bity ku celkovému počtu přijatých bitů [14]. Na obrázku 3.1 je zobrazena bitová chybovost bez použití kanálového kódování v závislosti na SNR. Použitím kódovacích metod a opravných kódů je možné chybovost snížit, což bude popsáno v následujících kapitolách. Pro jednotlivé kódy byla bitová chybovost počítána jako funkce SNR, kdy bylo pro dosažení co nejpřesnějších hodnot provedeno vždy sto tisíc iterací.

$$BER = \frac{počet\ chybných\ bitů}{počet\ všech\ bitů} \tag{3.1}$$

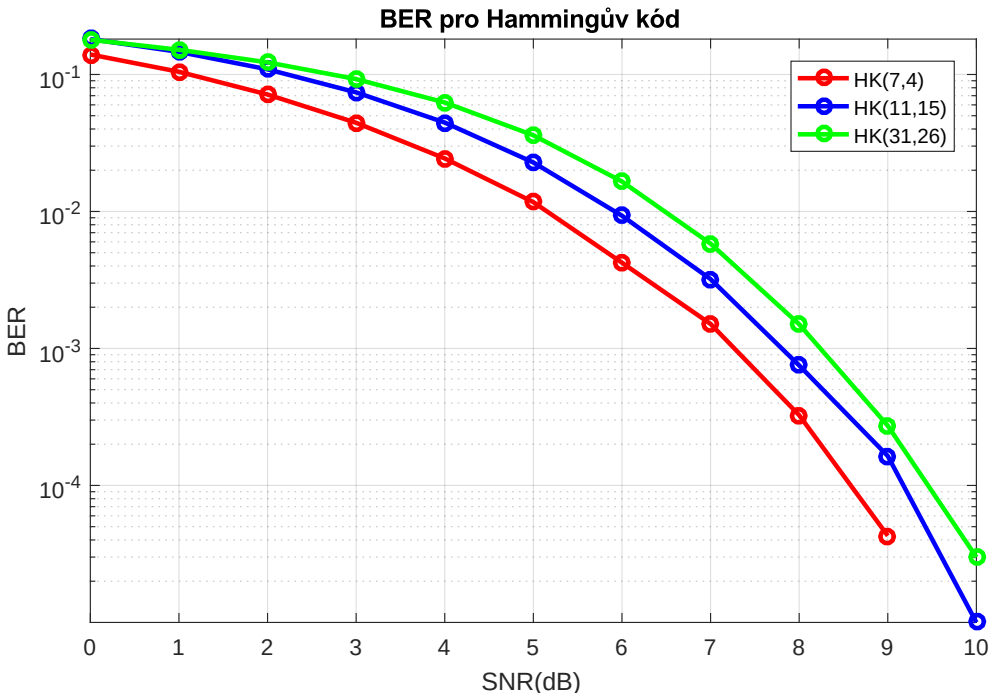


Obr. 3.1: Závislost chybovosti na SNR pro přenos nekódovaného slova.

3.1 BER Hammingův kód

Hammingův kód patří k základním a nejjednodušším kódům. Jeho schopností je detekce a oprava pouze jedné chyby. To znamená, že při nízké hodnotě SNR, kdy je chybou zasaženo více bitů, nemůže dojít ke korektní opravě. Pro dlouhé kódové slovo

dojde k ovlivnění více bitů a celková chybovost se tak zvýší. Srovnání pro jednotlivé kódové délky nalezneme na obrázku 3.2.



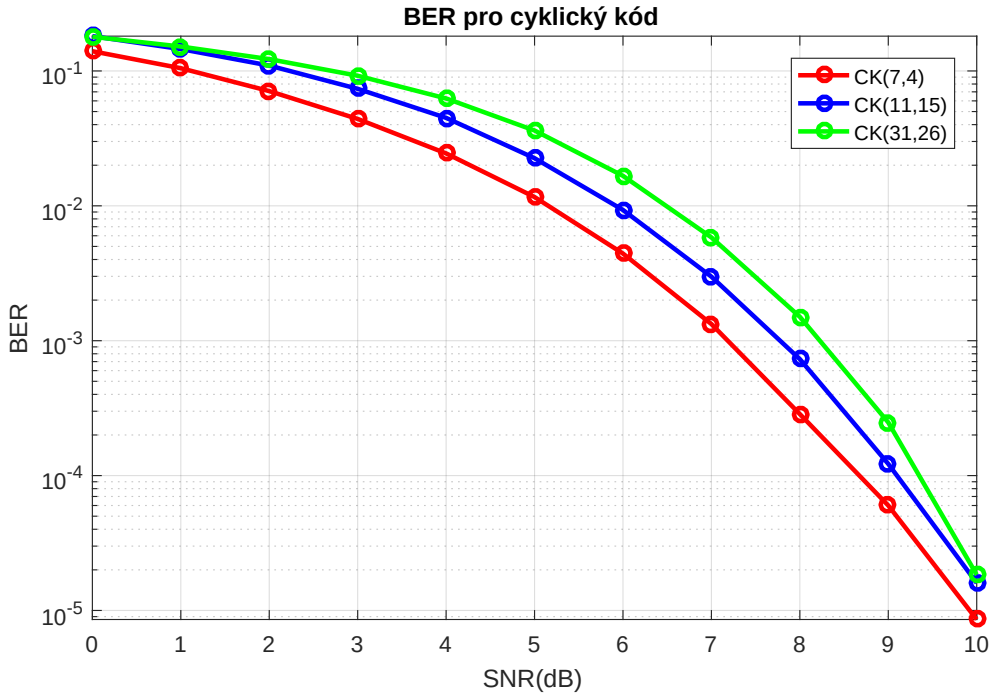
Obr. 3.2: Závislost chybovosti na SNR pro Hammingův kód.

3.2 BER cyklický kód

Cyklický kód se v praxi využívá spíše pouze k detekci chyby jako CRC doplněk a pokud je chyba indikována, vyžádá se nové zaslání slova. Jak však bylo ukázáno, pomocí posuvných registrů je možné provádět také opravu přijatého slova. Stejně jako Hammingův kód je však schopen opravit pouze jednonásobnou chybu, tudíž jeho účinnost není nejlepší a ve srovnání s Hammingovým kódem dosahují podobných výsledků, viz graf na obrázku 3.3.

3.3 BER konvoluční kód

Bitová chybovost konvolučních kódů v závislosti na SNR dále závisí na nastavení kódu. A to na rychlosti, tj. počtu vstupních bitů ku počtu bitů na výstupu, a délce informačního slova. Výsledná závislost je na obrázku 3.4. Zde je vidět, že nižší rychlost, tzn. větší počet bitů kódového slova, zajišťuje lepší BER, což je však vykoupeno větší náročností na zpracování, případně vyšší frekvenci při dékódování informace.



Obr. 3.3: Závislost chybovosti na SNR pro cyklický kód.

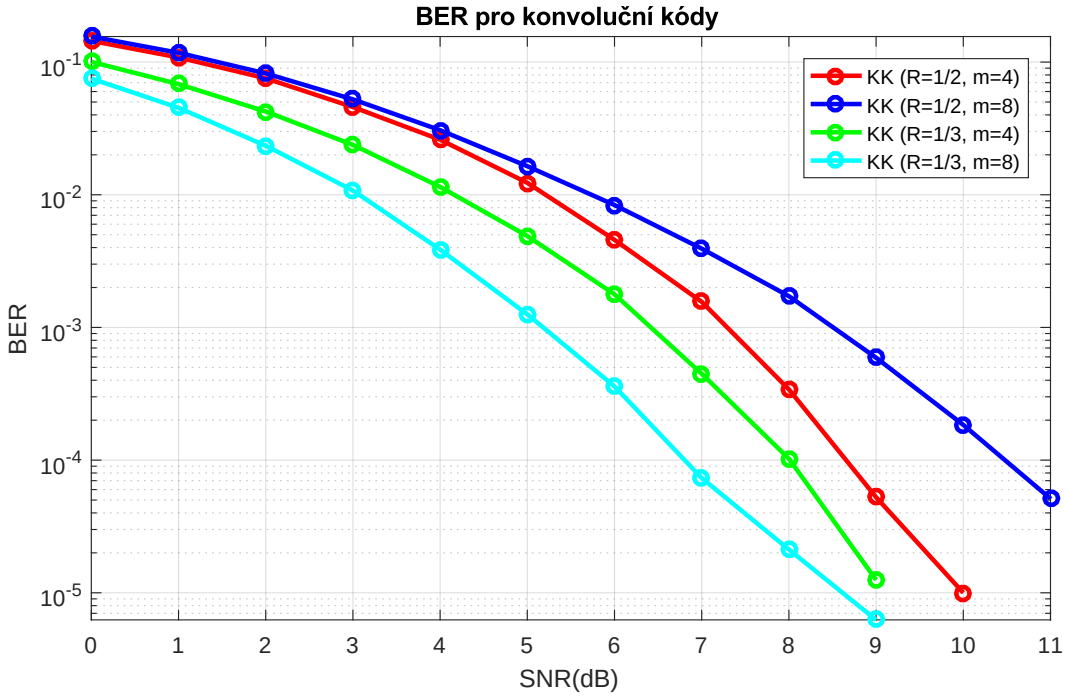
Jak je v grafu patrno, pro rychlost kódu $R = 1/3$ je dosaženo nižší chybovosti při delším informačním slově.

3.4 BER LDPC kód

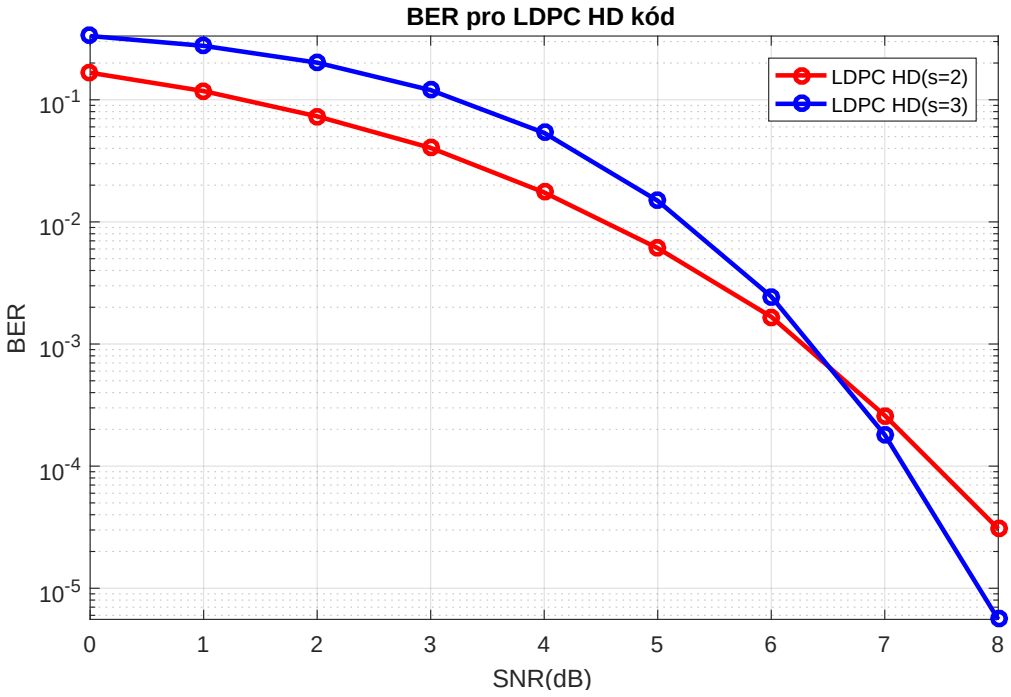
LDPC kódy dosahují z realizovaných kódů nejnižší bitové chybovosti. Nejlépe je na tom podle předpokladů algoritmus SP. Chybovost dále klesá s rostoucím řádem kódu, kdy se v praxi používají rozsáhlé matice. Pro názornost byly odsimulovány celkem tři naprogramované algoritmy pro řád $s = 2$ a 3. Ačkoli není algoritmus HD iterativní, při nízkých řádech si nevede špatně a je srovnatelný s ostatními metodami, jejichž účinnost výrazně roste s řádem a počtem iterací.

3.5 BER vzájemné srovnání všech použitých kódů

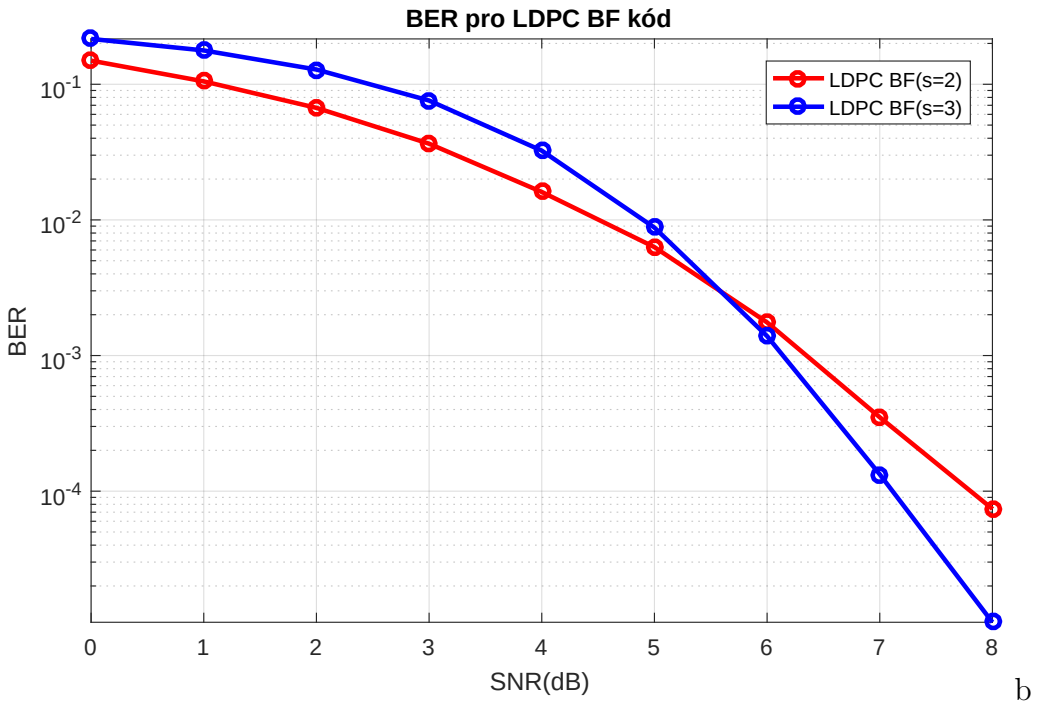
Chybovosti všech kódů byly popsány v předchozích kapitolách. Vychází z toho, že jak Hammingovy, tak cyklické kódy jsou na tom s chybovostí velice podobně, jelikož oba jsou schopné detekovat a opravit pouze jednu chybu. Konvoluční kód dosahuje o něco přijatelnějších výsledků. Nejlépe je na tom LDPC kód, jak je vidět také v grafu na obrázku 3.8, pro který je realizováno několik dekódovacích algoritmů, jež mají



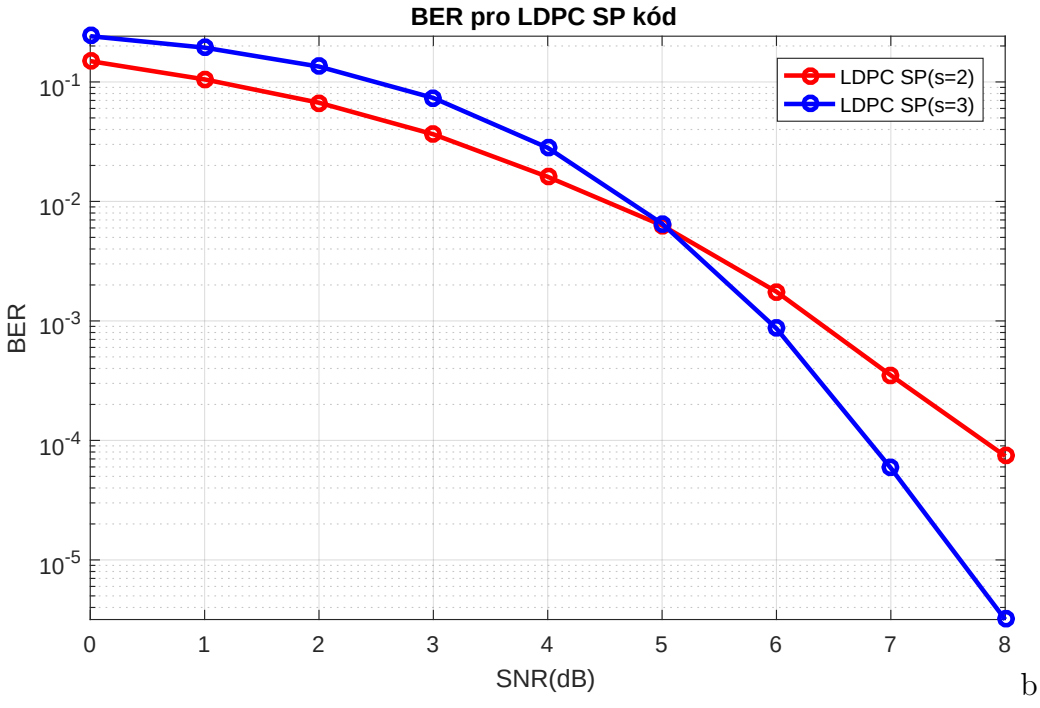
Obr. 3.4: Závislost chybovosti na SNR pro konvoluční kód.



Obr. 3.5: Závislost chybovosti na SNR pro LDPC HD kód.

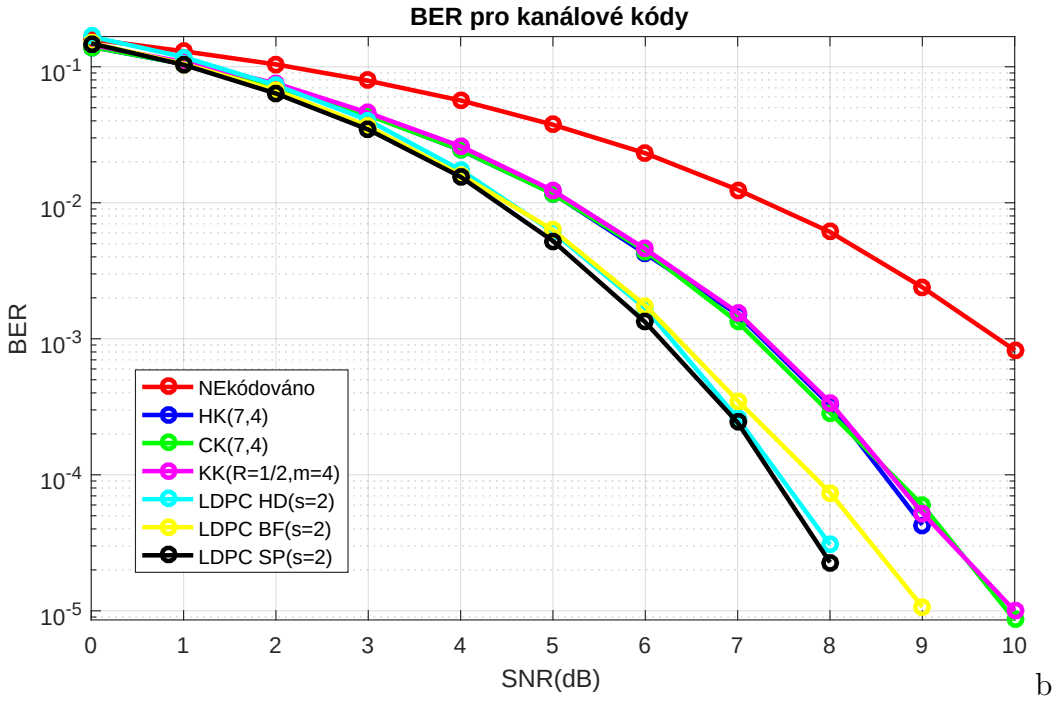


Obr. 3.6: Závislost chybovosti na SNR pro LDPC BF kód.



Obr. 3.7: Závislost chybovosti na SNR pro LDPC SP kód.

různou efektivitu. Pro pochopení a k výuce se používají krátké délky kódů, které nejsou tak efektivní, a proto v grafu nejsou tolik výrazné rozdíly mezi jednotlivými metodami. V praxi se však využívají kódy vyšších rozměrů, u nichž jsou rozdíly ve výsledné chybovosti markantnější.



b

Obr. 3.8: Závislost chybovosti na SNR pro vybrané kanálové kódy.

4 Návod na počítačové cvičení

Cílem práce je vytvoření výukového programu pro demonstraci kanálového kódování, a proto byl kladen důraz na intuitivní ovládání a názorné zobrazení celého procesu. Uživatelé mají možnost zadat libovolné vstupní data ve formě jednotlivý bitů, sledovat průběh kódování, simulovat průchod přenosovým kanálem a následně nahlédnout do procesu dekódování. Kompletní zadání počítačového cvičení je obsaženo v příloze A, zde je vypsáno pouze zadání úkolů k vypracování. V úvodu návodu na cvičení je shrnuta nutná teorie k jednotlivým kanálovým kódům, podle které programy pracují. Následuje zadání, jež studenti plní s pomocí vytvořeného programu do nachystaného protokolu s tabulkami. Na závěr je vyžadováno zhodnocení získaných poznatků.

Zadání

1. Každý z uvedených kódů si otevřete v simulačním programu a projděte si jejich funkci. Ověřte si znalosti kódování a dekódování. Vyzkoušejte si, kolik chyb je každý kód schopen korektně opravit.
2. Informační bity [1 1 0 1] zakódujte pomocí všech kódů a zaznamenejte do tabulky níže.
3. Bylo přijato kódové slovo [0 1 1 0 1 0 0] zakódované pomocí Hammingova kódu. Zjistěte, zda při přenosu došlo k chybě. Pokud ano, určete pozici chyby a opravte ji. Dekódujte informační bity.
4. Pomocí cyklického kódu zakódujte posloupnost informačních bitů [0 0 1 0 1 0 0 0 0 1 1] a určete hodnoty bitů paritních. Experimentálně ověřte, kolik chyb je pomocí cyklického kódu možné opravit.
5. Pomocí konvolučního kódu s rychlostí $R = 1/2$ zakódujte bity [1 0 1 0 0 1]. Vytvořte v kódové zprávě 2 chyby na pozicích 1 a 9 a stanovte, zda bylo přijatou zprávu možno opravit a dekódovat tak správně informační bity. Jaké byly celkové Hammingovy váhy ostatních cest?
6. S využitím LDPC kódu zakódujte slovo [0 1 0 1 1 0 0]. Vytvořte chyby na pozicích 5, 13, 14 a zjistěte, který z možných algoritmů je schopen chyby korektně opravit, případně kolik bylo potřeba iterací.
7. Seřaďte diskutované kódy podle účinnosti dekódování a opravy chyby.

Závěr

V úvodu práce byla popsána teorie k vybraným kanálovým kódům. Jedná se především o Hammingův, cyklický, konvoluční a LDPC kód. Každý z uvedených kódů byl podrobně rozebrán, byly popsány různé metody kódování a dekódování. Aby byla uvedená teorie názornější, byly ke každé metodě představeny příklady. Hammingův kód vychází z generující G a kontrolní H matice, které jsou využity ke kódování a dekódování. Metoda kódování a dekódování cyklického kódu je založena na obvodu z posuvných paměťových registrů, který je tvořen podle zadaného generujícího polynomu. Princip kodéru konvolučního kódu vychází z mřížového diagramu, jenž vede ke snadnějšímu pochopení. Dekódování je založeno na Viterbiho algoritmu, který prochází všechny možné cesty a vyhodnocuje nejpravděpodobnější cestu použitou ke kódování. Pro LDPC kódy je zásadní vytvoření generující a paritní matice, jež obsahují nízký počet prvků "1". Tomuto tématu jsou věnovány kapitoly 1.5.1 až 1.5.3. Ke kódování informace se využije generující matice. Pro proces dekódování byly aplikovány celkem tři algoritmy lišící se ve složitosti a účinnosti.

Druhá část práce vychází z uvedené teorie a popisuje praktickou část. V programovacím prostředí Matlab byly napsány funkce, které simulují uvedené kódy. Pro snadnější ovládání bylo navíc vytvořeno uživatelské rozhraní. V něm je možno zadat vstupní posloupnost bitů a definovat nutné parametry daného kódu. Poté lze informační slovo zakódovat a názorně sledovat tento proces. Do zakódovaného slova je umožněno vložit chybu, čímž se simuluje průchod komunikačním kanálem. Následné dekódování provede kontrolu správnosti přenosu a případně, pokud to lze, opraví přijaté slovo a dekóduje informační bity.

Ve třetí kapitole jsou shrnuty diskutované kódy z hlediska bitové chybovosti a vzájemně porovnány. Také je provedeno srovnání v závislosti na délce informačního slova, případně rychlosti konvolučního kódu nebo řádu LDPC. Získané výsledky jsou okomentovány a popsány.

Poslední kapitola se věnuje vytvoření zadání pro počítačové cvičení. V úvodu je uvedena nezbytná teorie k daným kódům, aby byl uživatel schopen porozumět výstupům programu a dát si je do souvislosti. Následuje zadání, které budou studenti plnit s využitím programu a výsledky zapisovat do připravených tabulek. Dosažené poznatky a výsledky bude vyžadováno zhodnotit v závěru protokolu.

Literatura

[1] PROKEŠ, Aleš. *Rádiové komunikační systémy.* Purkyňova 118, 612 00 Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2013. ISBN 978-xxxxxx.

[2] HUFFMAN, W. Cary a Vera PLESS. *Fundamentals of Error Correcting Codes.* Cambridge: Cambridge University Press, 2003. ISBN 978-0-511-078280-7.

[3] LIN, Shu a Daniel J. COSTELLO. *Error control coding: fundamentals and applications.* 2nd ed. Upper Saddle River, N.J.: Pearson-Prentice Hall, c2004. ISBN 01-304-2672-5.

[4] VLČEK, Karel. *Komprese a kódová zabezpečení v multimediálních komunikacích.* Praha: BEN - technická literatura, 2000. ISBN 80-860-5668-6.

[5] SWEENEY, Peter. *Error control coding: from theory to practice.* 2002. New York: Wiley, c2002. ISBN 04-708-4356-X.

[6] ELLIS, Robert L. a Israel GOHBERG. *Orthogonal Systems and Convolution Operators.* Springer Basel AG: Birkhăuser Verlag, 2003. ISBN 978-3-0348-9418-0.

[7] BENVENUTO, Nevio a Giovanni CHERUBINI. *Algorithms for communications systems and their applications.* 2002. New York: J. Wiley, 2002. ISBN 04-708-4389-6.

[8] SKLAR, Bernard. *Digital communications: fundamentals and applications.* 2nd ed. Upper Saddle River, N.J.: Prentice-Hall PTR, c2001. ISBN 01-308-4788-7.

[9] COOKE, Richard G. *Infinite matrices and sequence spaces.* London: Macmillan, 1950.

[10] MOREIRA, Jorge Castiñeira a Patrick Guy FARRELL. *Essentials of Error-Control Coding.* Chichester: Wiley, 2006. ISBN 978-0-470-02920-6.

[11] GALLAGER, Robert G. *Low Density Parity Check Codes.* 1963, 90 s, Dostupné z: http://www.inference.phy.cam.ac.uk/mackay/gallager/papers/

[12] D. J. C. MacKay and R. M. Neal *Near Shannon limit performance of low density parity check codes.* in Electronics Letters, vol. 33, no. 6, pp. 457-458, 13 March 1997.

[13] ODENWALDER, J. P. *Error Control Coding Handbook.* No. F44620- 76-C0056. San Diego: Linkabit Corporation, 1996.

[14] RAO, K. Deergha. *Channel Coding Techniques for Wireless Communications.* India: Springer, 2015. ISBN 978-81-322-2291-0.

[15] PETERSON, W. W. a E. J. WELDON. *Error-Correction Codes.* Second edition. Cambridge: MIT Press, 1972. ISBN 0 262 16 039 0.

Seznam symbolů, veličin a zkratek

A	amplituda
$AWGN$	Additive White Gaussian Noise – přídavný bílý gaussovský šum
BER	Bit Error Rate – bitová chybovost
$BPSK$	Binary Phase Shift Keying – dvoustavové fázové klíčování
BF	dekódovací algoritmus Bit – flipping
c	kódové slovo
CK	cyklický kód
c_p	přenesená zpráva
c_k	opravená zpráva
d	vektor opraveného slova
d	Hammingova vzdálenost
$d_{\min}$	minimální Hammingova vzdálenost
EG	euklidovská geometrie
f_j^x	odhad pravděpodobnosti přijatého symbolu
G	generující matice
G_{EG}	kontrolní matice LDPC kódu vytvořena pomocí euklidovské geometrie
GF	konečná tělesa – Galois Field
$g(x)$	generující polynom
H	kontrolní matice
HD	dekódovací algoritmus Hard Decision
HK	Hammingův kód
H_{EG}	paritní matice LDPC kódu vytvořena pomocí euklidovské geometrie
I_{k}	jednotková matice
J	počet řádků v H matici
k	počet informačních bitů
k_d	počet opravitelných chyb
k_{ol}	počet opravitelných chyb pro d_{min} liché
k_{os}	počet opravitelných chyb pro d_{min} sudé
$LDPC$	kód s řídkou paritní maticí – Low Density Parity Check
m	informační slovo
n	počet bitů zakódované zprávy
P	část matice zabezpečující kontrolu parity
p	paritní bity
p_x	polynomy
p^{r}	rozměr GF
q_{x}	podíl po dělení generujícím polynomem

Q_{ij}^{x}	funkce algoritmu Sum – Product
R	rychlost kódu
r	řád prvků GF
r	řídkost H matice LDPC kódu
r_{x}	zbytek po dělení
rx	přijaté kódové slovo
r_x	posuvné registry v blokovém schématu kodéru CK
R_{ij}^{x}	koeficienty algoritmu Sum – Product
S	kontrolní vektor dekódovacího algoritmus Bit – flipping
SP	dekódovací algoritmus Sum – Product
SNR	Signal to Noise Ratio – poměr signál/šum
$S1, S2$	spínače v blokovém schématu CK
s_x	posuvné registry v blokovém schématu dekodéru CK
s	syndrom
s	stupeň EG – LDPC kódu
U^{m}	některá z možných přijatých sekvencí
v	vektor pro tvorbu paritní H matice LDPC kódu
x	počet paritních symbolů
y_j	hodnota přijatého bitu na pozici j
Z	přijaté slovo
z_{x}	informační slovo posunuté doleva o n-k pozic
γ	počet jedniček na sloupec H matice LDPC kódu
ρ	počet jedniček na řádek H matice LDPC kódu

Seznam příloh

A Návod na počítačové cvičení

Teoretický úvod

Cílem počítačového cvičení je prohloubit znalosti z teorie kanálového kódování a s využitím simulačního programu ověřit nabyté vědomosti v praxi. Bude se jednat o kódy: Hammingovy, cyklické, konvoluční a LDPC.

Kanálové kódování zabezpečuje data proti chybám při přenosu. Tomuto kódování předchází zdrojové kódování. Jelikož vyslaná data prochází přenosovým kanálem, kde se vyskytuje rušení, může při přenosu dojít k záměně některých bitů. Proto byly vynalezeny kanálové kódy, které dokáži chybu nejen odhalit, ale do určité chybovosti podle typu kódu také opravit.

Hammingův kód

Tento kód patří k nejzákladnějším v teorii přenosu informace. Ke kódování a dekódování se využívají generující G a paritní H matice, jejichž odvození lze nalézt v odborné literatuře. Kódování informačních bitů poté probíhá vynásobením s generující G maticí dle následujícího vztahu:

$$c = m.G.$$

Dekódování se uskuteční vynásobením kódového slova s paritní H maticí. Výsledek násobení tzv. syndrom značí, zda se v přijatém slově vyskytla chyba. Pokud je syndrom nulový, k chybě nedošlo. V opačném případě, jelikož je Hammingův kód schopen opravit pouze jednonásobnou chybu, hodnota syndromu přímo značí pozici chyby v kódovém slově. Po opravě pouze stačí zpětně vyčíst informační bity.

Cyklický kód

Jak název napovídá, kódování i dekódování využívá v každém kroku cyklického posuvu. Proto je tento kód jednoduše implementovatelný s použitím kruhových posuvných registrů. Blokové schéma kodéru i dekodéru se tvoří z generujícího polynomu, jehož nejvyšší mocnina odpovídá počtu kontrolních bitů.

Jedná se o systematické kódování, tudíž kódové slovo obsahuje na nejvyšších pozicích informační bity a k nim se připojí daný počet bitů kontrolních. Ty jsou tvořeny z obsahu buněk posuvných registrů po průchodu všech informačních bitů.

K procesu dekódování je obvod sestaven opět z posuvných registrů, paměti a sčítaček. Pokud při přenosu nedošlo k chybě v kódovém slově, jsou po nasunutí všech bitů přijaté zprávy obsahy všech posuvných registrů nulové. Pokud tomu tak není, následuje druhý krok opravy, kdy se na vstup obvodu přivádějí bity s nulovou hodnotou a v obvodu sčítačky se generuje opravný bit, který se sčítá s přijatým slovem. Pomoci cyklického kódu je možno opravit pouze jednonásobnou chybu. Takže pokud nalezneme opravný bit s hodnotou "1", můžeme dekódování ukončit a zbytek

bitů přijatého slova vyčíst rovnou z paměti. Na závěr vyčteme požadovaný počet informačních bitů z nejvíce významných pozic kódového slova.

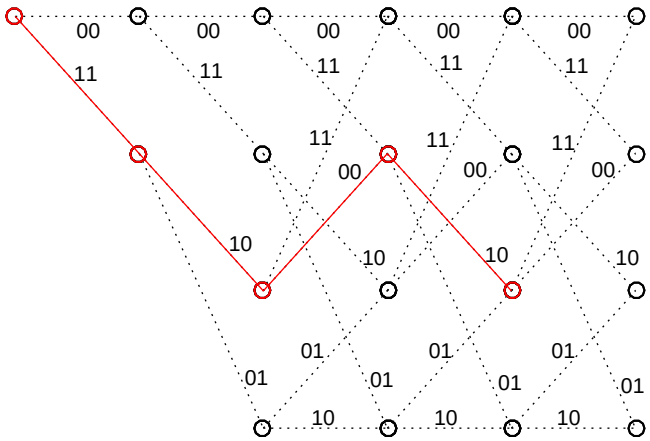
Konvoluční kód

Doposud popisované kódy spadaly do kategorie blokových. To znamená, že se daný kód aplikuje na přesný počet informačních bitů, přičemž se kódy označují jako bez paměti. Na rozdíl od toho je konvoluční kód označován jako kód s pamětí, jelikož výstupní hodnota nezávisí pouze na vstupním bitu, ale také na několika předchozích.

Konvoluční kódování využívá tzv. trellis (mřížový) diagram. Ten je sestaven z generujícího polynomu, určujícího navíc tzv. rychlost kódu R, což je poměr mezi vstupními a výstupními bity na jeden znak zprávy.

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

Kódování probíhá pohybem v mřížovém diagramu. Pokud kódujeme bit "0", jdeme horní cestou a v případě bitu "1" spodní. Kódové slovo získáme seřazením hodnot příslušejících jednotlivým cestám, viz obrázek níže, pro informační bity [1 0 1 0] získáme kódové slovo [1 1 1 0 0 0 1 0].



Obr. A.1: Mřížový (trellis) diagram se zvýrazněním cesty pro kódování.

Dekódování probíhá nejčastěji tak, že se porovná přijatá zpráva, která může být zasažena chybou se všemi posloupnostmi, jež mohly být vyslány. Hledá se tzv. věrohodnostní poměr a snažíme se o jeho maximalizaci. Pokud je přijetí všech kódových slov stejně pravděpodobné, dekodér který dosahuje nejlepšího věrohodnostního poměru je ten, který porovnává podmíněné pravděpodobnosti. *A. J. Viterbi* vyvinul velice efektivní metodu dekódování založenou na maximalizaci věrohodnostního poměru. Dekódování spočívá v porovnání přijaté zprávy se všemi možnými cestami v mřížovém diagramu. Zároveň se pro jednotlivé cesty počítá Hammingova vzdálenost, tj. rozdíl mezi přijatým a možným kódovým slovem. Jako nejpravděpodobnější

kódové slovo se nakonec určí to, jehož cesta mřížovým diagramem má nejmenší celkovou Hammingovu vzdálenost od přijatého slova. Pokud je celková Hammingova vzdálenost rovná nule, bylo pravděpodobně přijato právě vyslané kódové slovo.

Postup dekódování je rozdělen do několika bodů:

- nejdříve se přijatá zpráva rozdělí na bloky po n bitech, kdy každý blok vyjadřuje jeden informační bit.
- poté se bloky rozepíší nad jednotlivé úseky v mřížovém diagramu a porovnávají se se všemi možnými cestami, přičemž se zaznamenává vzájemná Hammingova váha, tj. počet odlišností, které se iterativně sčítají.
- v prvním úseku jsou pouze dvě možné cesty, v druhém čtyři a v každém dalším osm.
- jelikož se v každém dalším kroku sejdou v jednom ze čtyř uzlů dvě možné cesty, vybere a zachová se z nich pouze ta, která má menší Hammingovu vzdálenost vůči přijaté zprávě. Pokud by měly dvě cesty stejnou Hammingovu váhu, vybere se náhodně jedna z nich.
- na konci dekódování se určí jako nejpravděpodobnější cesta (vyslané slovo) ta, která má nejmenší celkovou Hammingovu váhu.
- pokud by došlo k případu, což se může stát, že na konci dekódování mají stejnou Hammingovu váhu dvě nebo více cest, nelze vyslané slovo jednoznačně určit.

Low Density Parity Check- LDPC kód

K dekódování se využívá kontrolní H matice, která obsahuje nízký počet "1" jak v řádcích, tak ve sloupcích a ostatní prvky jsou hodnoty "0". Odtud pochází název *kód s řídkou paritní maticí*. Způsob vzniku generujících a paritních matic vychází například z euklidovské geometrie a princip jejich vytvoření je popsán v odborné literatuře. Kódování je poměrně jednoduché a vychází opět z násobení informačních bitů s generující maticí stejně jako v případě Hammingova kódu.

Dékódování je již komplexnější proces a bylo vytvořeno několik algoritmů lišících se nejen složitostí, ale také účinností. V simulačním programu je možné vybrat celkem ze tří možností, jejichž princip je popsán níže:

Dékódovací algoritmus **Hard decision**: jedná se o neiterativní algoritmus skládající se z vertikálního a horizontálního kroku. K dekódování vedou následující kroky:

- v prvním kroku si vypíšeme tabulku pozic podle paritní matice, na kterých je hodnota "1" a k těmto pozicím přiřadíme bit z přijatého slova ze stejné pozice.
- v kroku druhém opět nahlédneme do paritní matice a vytvoříme novou tabulku. Její řádky se budou skládat z pozic v jednotlivých sloupcích, na kterých je hodnota "1". Dále budeme pracovat s tabulkou vytvořenou v prvním kroku a počítat nové hodnoty řádků. Pro první řádek označovaný jako c_1 (první sloupec v paritní matici H_{EG}) vypočítáme hodnotu prvního sloupce jako sou-

čet modulo(2) z řádku v první tabulce na pozici odpovídající první jedničce v prvním sloupci paritní matice, vyjma aktuálně počítanou pozici.
- v posledním kroku připojíme na konec tabulky přijaté slovo a podle majoritní logiky se rozhodne o hodnotě každého bitu kódového slova.

V programu jsou k dispozici obě tabulky, podle kterých si můžete ověřit funkci algoritmu.

Bit-flipping: tento algoritmus je iterativní. Na začátku se počítá syndrom přijatého slova s pomocí paritní H matice a pokud je syndrom nulový, bylo přijato právě vyslané slovo a není potřeba provádět opravu. Pokud je však alespoň jeden bit ve vypočteném syndromu hodnoty "1", byla detekována chyba při přenosu a algoritmus se pokusí o její opravu. Při opravě se počítá s vypočteným syndromem s, který je násoben se sloupci paritní matice a vznikne vektor S. Následně se porovnává, zda je hodnota vektoru S na pozici S_i větší, než předchozí největší hodnota. Pokud podmínka platí, pozice se uloží a přijaté bity se na těchto hodnotách na závěr negují. Znovu se vypočítá syndrom s a v případě, že není nulový, se algoritmus opakuje. Postup je následující:

- vypočteme hodnotu syndromu mezi přijatým slovem a kontrolní H_{EG} maticí.
- pokud není syndrom na všech pozicích nulový, vypočítáme pomocný vektor S podle vzorce

$$S = s.H.$$

- porovnáme hodnoty vektoru S a v případě, že hodnota na pozici S_i je větší než, předchozí největší hodnota, uložíme pozici i (hodnotám i odpovídá pro první prvek vektoru S hodnota "1", až poslednímu prvku vektoru hodnota "2^{2s}").
- negujeme hodnoty v přijatém slově na pozicích určených v předchozím kroku.
- vypočteme nový syndrom s a pokud není na všech pozicích nulový, opakujeme algoritmus až do maximálního zadaného počtu iterací.

V programu jsou vypsány všechny důležité hodnoty jako syndrom, vektor S, pozice pro negaci bitů a samotné opravené slovo.

Sum-product: iterativní algoritmus s měkkým rozhodováním využívající simulaci modelu kanálu pomocí přídavného šumu. K opravě přijatého slova dochází na základě výpočtu pravděpodobnosti, že se přijatý bit rovná "1" nebo "0". Jelikož se jedná o poměrně složitý výpočet, jsou v programu zobrazeny pouze zmíněné pravděpodobnosti a zbytek postupu je pouze nastíněn.

Zadání

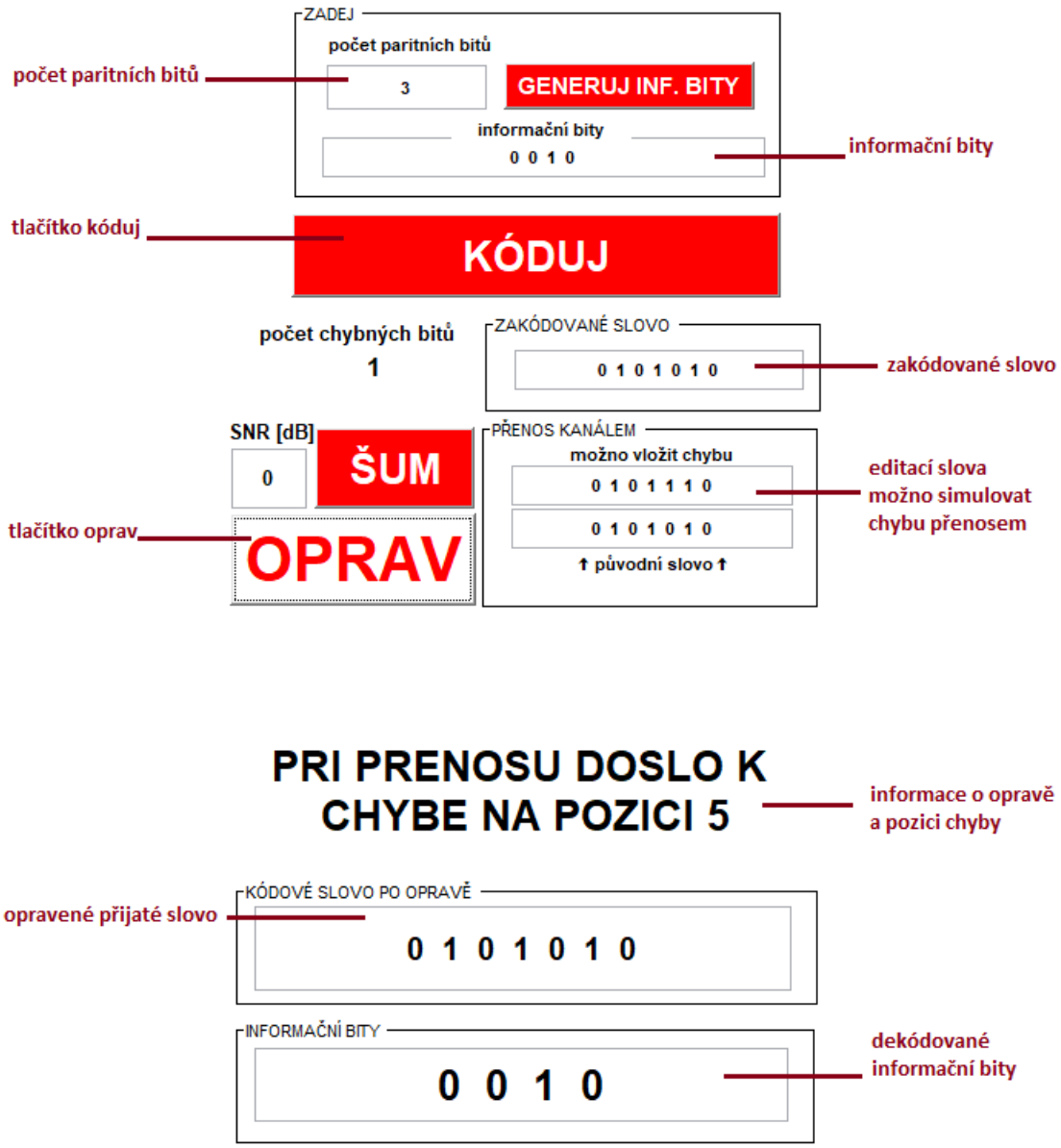
1. Každý z uvedených kódů si otevřete v simulačním programu a projděte si jejich funkci. Ověřte si znalosti kódování a dekódování. Vyzkoušejte si, kolik chyb je každý kód schopen korektně opravit.
2. Informační bity [1 1 0 1] zakódujte pomocí všech kódů a zaznamenejte do tabulky níže.
3. Bylo přijato kódové slovo [0 1 1 0 1 0 0] zakódované pomocí Hammingova kódu. Zjistěte, zda při přenosu došlo k chybě. Pokud ano, určete pozici chyby a opravte ji. Dekódujte informační bity.
4. Pomocí cyklického kódu zakódujte posloupnost informačních bitů [0 0 1 0 1 0 0 0 0 1 1] a určete hodnoty bitů paritních. Experimentálně ověřte, kolik chyb je pomocí cyklického kódu možné opravit.
5. Pomocí konvolučního kódu s rychlostí $R = 1/2$ zakódujte bity [1 0 1 0 0 1]. Vytvořte v kódové zprávě 2 chyby na pozicích 1 a 9 a stanovte, zda bylo přijatou zprávu možno opravit a dekódovat tak správně informační bity. Jaké byly celkové Hammingovy váhy ostatních cest?
6. S využitím LDPC kódu zakódujte slovo [0 1 0 1 1 0 0]. Vytvořte chyby na pozicích 5, 13, 14 a zjistěte, který z možných algoritmů je schopen chyby korektně opravit, případně kolik bylo potřeba iterací.
7. Seřaďte diskutované kódy podle účinnosti dekódování a opravy chyby.

Pokyny pro vypracování

Spuštěním programu "Kanálové kódy", se Vám zobrazí okno s nabídkou 4 kódů. Vyberte požadovaný kód stiskem příslušného tlačítka. Rozložení uživatelského rozhraní je pro všechny kódy podobné, liší se pouze v zadávání specifických parametrů daného kódu a zobrazením procesů kódování/dekódování. V horní části programu se zadají informační bity a požadované parametry, následným stiskem tlačítka "**KÓDUJ**" dojde k zakódování informačních bitů. Proces kódování můžete sledovat v pravé části programu. Zakódované informační bity jsou v poli "**ZAKÓDOVANÉ SLOVO**".

Chybu do přeneseného slova vnesete editací pole "**PŘENOS KANÁLEM**". Následným stiskem tlačítka "**OPRAV**" dojde k opravě a dekódování informačních bitů. Opravené kódové slovo naleznete v levé spodní části programu v poli "**KÓDOVÉ SLOVO PO OPRAVĚ**" a dekódované informační bity jsou o políčko níž. O průběhu opravy jste informování na obrazovce programu a průběh dekódování je možné sledovat v pravé části okna. V případě nestandardních situací jste upozorněni vyskakovacím oknem.

Na následujícím obrázku je zobrazen náhled části okna Hammingova kódu s popisem ovládání, které je pro všechny kódy identické.



Obr. A.2: Náhled části okna programu Hammingova kódu s popisem ovládání.

Vypracování

Ad. 2 : Zakódujte informační bity

Zakódujte informační bity [1 1 0 1]	
Hammingův kód	
Cyklický kód	
Konvoluční kód	
LDPC kód	

Ad. 3 : Cvičení na Hammingův kód

Přijaté slovo [0 1 1 0 1 0 0]	———
Při přenosu došlo k chybě	ANO \ NE
Pozice chyby	
Opravené kódové slovo	
Informační bity	

Ad. 4 : Cvičení na cyklický kód

Zakódujte informační bity [0 0 1 0 1 0 0 0 0 1 1]	———
Zakódované slovo	
Paritní bity	
Počet opravitelných chyb	

Ad. 5 : Cvičení na konvoluční kód

Zakódujte informační bity [1 0 1 0 0 1]	———
Zakódované slovo	
Je možné opravit kódové slovo	ANO \ NE
Hammingovy váhy ostatních cest	

Ad. 6 : Cvičení na LDPC kód

Zakódujte informační bity [0 1 0 1 1 0 0]	———
Zakódované slovo	
Které z algoritmů opraví poškozené slovo	HD \ BF \ SP
Počet iterací na opravu daným algoritmem	

Ad. 7 : Seřazení diskutovaných kódů podle účinnosti a počtu opravitelných chyb

Seřaďte kódy	———
1. (nejlepší)	
2.	
3.	
4.	

Zhodnocení výsledků, shrnutí poznatků

B Přehled jednotlivých m-funkcí

V této části přílohy jsou vypsány m-funkce pro jednotlivé kódy se stručným popisem. Hlavní okno programu se aktivuje spuštěním funkce "Program_pro_demonstraci-_kanaloveho_kodovani".m a následně se stiskem příslušného tlačítka volí požadovaný kód.

Tab. B.1: Výpis m-funkcí pro Hammingův kód a stručný popis.

Složka- Kanalove_kodovani_DP	
Program_pro_demonstraci_kanaloveho_kodovani.m	Spuštení programu pro demonstraci kanálového kódování
Složka- Hamminguv_kod	
Hamminguv_kod.m	Hl. soubor s GUI Hammingova kódu
generovani_matic.m	Generování G a H matice
nasobeni_matic_XOR.m	Násobení dvou matic v modulo(2) logice
kontrola_binar.m	Testování vstupních dat
zasumeni.m	Simulace průchodu přenosovým kanálem

Tab. B.2: Výpis m-funkcí pro cyklický kód a stručný popis.

Složka- Kanalove_kodovani_DP	
Program_pro_demonstraci_kanaloveho_kodovani.m	Spuštení programu pro demonstraci kanálového kódování
Složka- Cyklicky_kod	
Cyklicky_kod.m	Hl. soubor s GUI cyklického kódu
koder_CK.m	Kodér cyklického kódu
dekoder_CK.m	Dekodér cyklického kódu
kontrola_binar.m	Testování vstupních dat
zasumeni.m	Simulace průchodu přenosovým kanálem

Tab. B.3: Výpis m-funkcí pro konvoluční kód a stručný popis.

Složka- Kanalove_kodovani_DP	
Program_pro_demonstraci_kanaloveho_kodovani.m	Spuštení programu pro demonstraci kanálového kódování
Složka- Konvolucni_kod	
Konvolucni_kod.m	Hl. soubor s GUI konvolučního kódu
zasumeni.m	Simulace průchodu přenosovým kanálem
viterbi_dekod_graf3.m	Graf mřížového diagramu pro R=1/3 pro dekódování
viterbi_dekod_graf.m	Graf mřížového diagramu pro R=1/2 pro dekódování
trelis_kodovani_graf3.m	Graf mřížového diagramu pro R=1/3 pro kódování
trelis_kodovani_graf.m	Graf mřížového diagramu pro R=1/2 pro kódování
preskladani_matice_pro_cestu.m	Seřazení matice při dekódování sestupně podle uzlů pro vykreslení všech možných cest v mřížovém diagramu
preskladani_matice.m	Seřazení matice při dekódování sestupně podle uzlů pro vypsání hammingových vzdáleností jednotlivých cest
krokovani_dekod.m	Krokování dekódování v mřížovém diagramu
krokovani3.m	Krokování kódování v mřížovém diagramu pro R=1/3
krokovani.m	Krokování kódování v mřížovém diagramu pro R=1/2
kontrola_binar.m	Testování vstupních dat
koder_viterbi3.m	Kodér konvolučního kódu pro R=1/3
koder_viterbi.m	Kodér konvolučního kódu pro R=1/2
dekoder_viterbi3.m	Dekodér konvolučního kódu pro R=1/3, výsledkem je matice o osmi řádcích s Ham. váhou jednotlivých cest
dekoder_viterbi23.m	Dokončení dekódování pro R=1/3, vyhodnocuje nejpravděpodobnější cestu

dekoder_viterbi.m	Dekodér konvolučního kódu pro R=1/2, výsledkem je matice o osmi řádcích s Ham. váhou jednotlivých cest
dekoder_viterbi2.m	Dokončení dekódování pro R=1/2, vyhodnocuje nejpravděpodobnější cestu
dekod_inf_bitu3.m	Vyčtení informačních bitů z nejpravděpodobnější cesty pro R=1/3
dekod_inf_bitu.m	Vyčtení informačních bitů z nejpravděpodobnější cesty pro R=1/2
cista_mrizka_R1_3.m	Prázdná mřížka konvolučního kódu pro R=1/3 (kódování)
cista_mrizka.m	Prázdná mřížka konvolučního kódu pro R=1/2 (kódování)
cista_mrizka_dekod3.m	Prázdná mřížka konvolučního kódu pro R=1/3 (dekódování)
cista_mrizka_dekod.m	Prázdná mřížka konvolučního kódu pro R=1/2 (dekódování)
animace3.m	Animace kódování pro R=1/3
animace.m	Animace kódování pro R=1/2

Tab. B.4: Výpis m-funkcí pro LDPC kód a stručný popis.

Složka- Kanalove_kodovani_DP	
Program_pro_demonstraci_kanaloveho_kodovani.m	Spuštení programu pro demonstraci kanálového kódování
Složka- Cyklicky_kod	
LDPC_kod.m	Hl. soubor s GUI LDPC kódu
EG_LDPC.m	Generování G a H matic euklidovskou geometrií
nasobeni_matic_XOR.m	Násobení dvou matic v modulo(2) logice
kontrola_binar.m	Testování vstupních dat
zasumeni.m	Simulace průchodu přenosovým kanálem
oprava_LDPC_HD.m	Dekódovací algoritmus Hard Decision
oprava_LDPC_BF.m	Dekódovací algoritmus Bit Flipping
oprava_LDPC_SP.m	Dekódovací algoritmus Sum Product