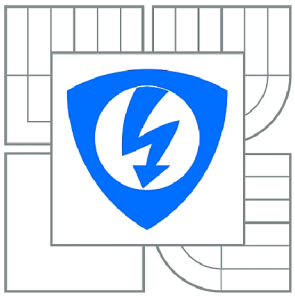
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

VIZUALIZACE PRINCIPŮ PROTICHYBOVÉHO ZABEZPEČENÍ BLOKOVÝMI A CYKLICKÝMI KÓDY

VISUALIZATION OF FORWARD ERROR CORRECTION SECURING BY BLOCK AND CYCLICAL CODES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

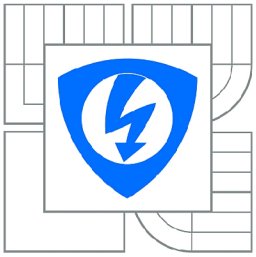
AUTOR PRÁCE — PETR FANČAL

AUTHOR

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

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Petr Fančal ***ID:*** 155111
Ročník: 3 ***Akademický rok:*** 2014/2015

NÁZEV TÉMATU:

Vizualizace principů protichybového zabezpečení blokovými a cyklickými kódy

POKYNY PRO VYPRACOVÁNÍ:

V prostředí programu Matlab realizujte výukovou aplikaci demonstrující principy zabezpečení blokovými a cyklickými kódy. Aplikace bude realizována pomocí grafického rozhraní GUI a bude krok za krokem vizualizovat dílčí části výpočtů při zabezpečení, detekci a opravě chyb. Aplikace rovněž umožní určit opravné schopnosti kódů, které bude možno vybírat z předdefinovaných či přímo zadávat. Aplikaci koncipujte tak, aby ji bylo možno snadno rozšiřovat, dostatečně doplněnou komentáři v kódech a nápovědou v programu. Ošetřete chybné zadání či volbu.

DOPORUČENÁ LITERATURA:

[1] Zaplatílek, K., Doňar, B. MATLAB: tvorba uživatelských aplikací. Praha: BEN, 2004. ISBN 80-7300-133-0.
[2] Morelos-Zaragoza, R. The art of error correcting coding. Chichester: John Wiley & Sons, 2002. ISBN 0-471-49581-6.
[3] Lin, S., Costello, D. J.. Error Control Coding: Fundamentals and Applications, Prentice Hall: Englewood Cliffs, NJ, 2005 ISBN: 0-13-042672-5.
[4] Moon, T. K. Error Correction Coding: Mathematical Methods and Algorithms. Wiley-Interscience, 2005, ISBN-13: 978-0070010697.

Termín zadání: 9.2.2015 ***Termín odevzdání:*** 2.6.2015

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

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

ABSTRAKT

Bakalářská práce seznamuje s principy protichybového zabezpečení lineárními blokovými a cyklickými systematickými kódy a jejich vizualizace pomocí výukového programu.
První část práce obsahuje ucelený teoretický základ principů protichybového zabezpečení a vlastnosti lineárních blokových a cyklických systematických kódů.
Druhá část popisuje samotný výukový program, realizovaný v grafickém prostředí GUI programu MATLAB. Tato aplikace krok za krokem vizualizuje dílčí části výpočtů při zabezpečení, detekci i opravě chyb. Jejím hlavním přínosem je využití jako názorné výukové pomůcky.

KLÍČOVÁ SLOVA

protichybový, lineární blokový kód, cyklický kód, MATLAB GUI, výukový program

ABSTRACT

This bachelor thesis provides an introduction to the principles of forward error correction securing by linear block and cyclical systematic codes and its visualization by educational application.
First part of the thesis contains comprehensive theoretical base of principles of forward error correction securing and capability of linear block and cyclical systematic codes.
Second part describes the educational application, realized in graphical user interface GUI of MATLAB. This application step by step visualizes parts of procedures of securing, error detection and correction. Its main benefit is its usability as illustrative teaching aid.

KEYWORDS

forward error correction, linear block code, cyclic code, MATLAB GUI, educational application

FANČAL, P. *Vizualizace principů protichybového zabezpečení blokovými a cyklickými kódy.*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 43 s. Vedoucí práce byl Ing. Pavel Šilhavý, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vizualizace principů protichybového zabezpečení blokovými a cyklickými kódy.“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

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

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Již v polovině 19.století s rozvojem informačních systémů vznikla nutnost přenosu dat a jeho optimalizace. Teorie přenosu informace, jejíž základní kameny položil C. E. Shannon svým modelem komunikačního procesu, umožnila vznik a vývoj kódů pro účelnou úpravu přenášené informace. Přenosové cesty a celé sítě se v následujících desetiletích rychle rozšiřovaly a vznikla celá řada kódů se specifickými vlastnostmi, vhodnými pro metalické, optické, bezdrátové i satelitní kanály.

Hlavními cíli kódování informace jsou především zrychlení přenosu odstraněním nadbytečnosti a tím lepšího využití přenosového kanálu, utajení dat proti odposlechu pomocí šifrování, ale především zajištění bezchybného přenosu dat detekcí nebo opravou chyb vzniklých přenosem prostředím s rušivými vlivy.

Lineární blokové a cyklické kódy systematické jsou základní součástí rodiny těchto protichybových kódů.

Tato bakalářská práce se zabývá především základním pochopením principu protichybového zabezpečení s využitím lineární algebry. V teoretické části si čtenář osvětlí své znalosti protichybového kódování i principu lineárního a cyklického systematického kódu a pomocí demonstrační aplikace si názorně prakticky otestuje jeho funkčnost v různých situacích. Pro zachování názornosti a přehlednosti procesů se demonstrační algoritmus zabývá pouze informačními prvky v binární soustavě. Tato práce se zabývá lineárními blokovými a cyklickými kódy systematickými.

1 TEORIE PROTICHYBOVÉHO KÓDOVÁNÍ

Při přenosu dat přes obecný komunikační kanál mezi zdrojem a příjemcem zprávy je zcela podstatná bezchybnost přenosu informace. Při přenosu informace vyjádřené datovou zprávou jakýmkoliv prostředím dochází k modifikaci původní zprávy rušivými vlivy tohoto prostředí a dochází tedy k chybám.

Úkolem procesu protichybového kódování je tyto chyby detekovat a umožnit tím opakování přenosu,nebo u výkonnějších kódů detekovat i opravit určitý počet chyb dle schopností použitého kódu. Chyby, které zabezpečovací kód není schopen již zjistit, přebírají k detekci i opravě vyšší vrstvy komunikačního spojení.

1.1 Principy protichybového kódování

1.1.1 Princip vzniku chyb

Při přenosu datového signálu prostředím dochází k interakci signálu s ostatními rušivými signály. Žádné reálné prostředí není bez těchto signálů, ať už se jedná o metalické vedení, optické nebo nejvíce zarušené spojení volným prostorem - tzv. bezdrátové spojení, do kterého zahrnujeme i satelitní spojení. Součet těchto rušivých signálů označujeme pojmem šum.

Datový signál v médiu (prostředí) lze rozpoznat a využít k přenosu zprávy pouze když není pohlcen přítomným šumem. Odstup úrovně výkonu signálu od úrovně výkonu šumu nazýváme SNR (Signal to Noise Ratio), který je podstatný pro realizaci datového přenosu pomocí signálu. Odstup šumu v daném prostředí dokážeme zvýšit pouze zvýšením vysílacího výkonu, který ale vytváří rušivé vlivy na blízké přenosové cesty a dochází k tzv. přeslechům. V praxi se vychází z doporučení ITU-T (ITU Telecommunication Standardization Sector is one of the three sectors of the International Telecommunication Union (ITU)), kde jsou stanoveny doporučené hodnoty vysílacích výkonů pro různé přenosové systémy.

Šum se svou vyrovnanou výkonovou úrovní není jediným zdrojem rušení v médiu. Jsou přítomny i rušivé signály, tzv. interferenční, které mají svůj časový průběh. Celkový součet šumu i rušení nazýváme souhrnně „hlukem“.

Při vysílání zprávy dochází k procesu kvantování, kdy vysílač určuje rozhodovací úroveň vyjádřenou veličinou použitého signálu (napětím). Přijímač naopak tuto hladinu použije pro získání informace a dekóduje přijatou zprávu. Pokud hluk přesáhne rozhodovací úroveň, pozmění původní informační hodnotu signálu a přijímač vyhodnotí v daném vzorkovacím okamžiku jinou úroveň a dojde k chybnému vyjádření prvku zprávy - vznikne chyba.

Chyba se projeví inverzí některých bitů datového přenosu a proto pro modelování vzniku chyby použijeme sčítačku modulo2.

Operace sčítání modulo 2 označujeme znakem $\oplus$ a jedná se o logickou operaci XOR (exclusive OR - logický součet odpovídající operaci modulo 2) :
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$

Sčítačku mod 2 můžeme vyjádřit vztahem :

$$J(x) = F(x) \oplus E(x) \tag{1.1}$$

kde $F(x)$ je zdrojová zpráva v binární formě a $J(x)$ je přenesená zpráva na vstupu přijímače. $E(x)$ je tzv. chybový polynom, který vyjadřuje modifikaci zdrojové zprávy. Pokud je tedy $E(x) = 0$ tak $F(x) = J(x)$.

Uvedené posloupnosti bitů lze vyjádřit vektorem, např. $\boldsymbol{f} = [1010101101]$ nebo velmi často polynomem

$$F(x) = a_0.x^0 + a_1.x^1 + a_2.x^2 ... + a_n.x^n \tag{1.2}$$

tedy vlastně rozkladem čísla ve dvojkové soustavě, při zápisu tedy zůstávají jen nenulové prvky. Například chybový vektor :

$$\boldsymbol{e} = [00101]$$

lze zapsat jako

$$E(x) = x^2 + x^4$$

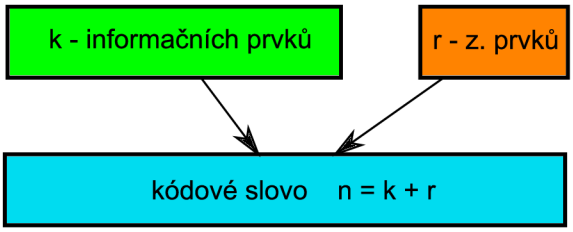
Chybovost je velmi důležitým parametrem reálného komunikačního spoje a lze popsat různými statistickými hodnotami, které se v praxi řídí podle revidovaného doporučení ITU-T G.826 a dalších norem. Měření chybovosti se provádí za provozu, některá celková měření lze však provést pouze v testovacím režimu spoje. Nejdůležitějším parametrem je bitová chybovost BER , při měření bit po bitu a je dána vztahem :

$$BER = \frac{n_{ch}}{n_c} \tag{1.3}$$

kde n_{ch} je počet chybných bitů a n_c je celkový počet bitů v měřeném úseku. V praxi hodnota BER nesmí klesnout pod hodnotu 10^{-3} , běžné provozní hodnoty se pohybují kolem 10^{-7} i méně.

1.1.2 Koncepce protichybových kódů

Protichybové kódy jsou založeny na vložení nadbytečné informace, která umožňuje rozlišit chybové kódové slovo od originálního (viz obrázek 1.1). Výsledné kódové slovo má různý tvar, podle metodiky kódovacího procesu, a jeho celkovou délku označujeme písmenem n. Počet redundantních prvků označujeme r a nakonec počet prvků představujících původní zprávu označujeme k. Kódy většinou zapisujeme na základě těchto parametrů jako : *Název (n ; k)*, nebo výstižněji *Název (n ; k ; d_{min})*.



Obr. 1.1: Koncepce protichybových kódů

Nejpodstatnějším parametrem kódu je jeho *kódový poměr* R (code rate), který je určen poměrem :

$$R = \frac{k}{n} \tag{1.4}$$

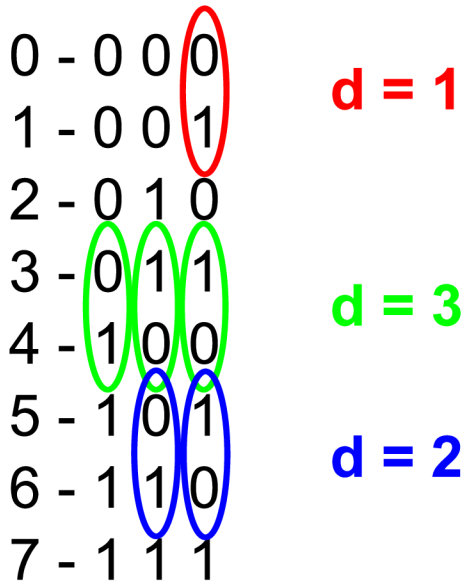
Tento parametr není možné volit libovolně, neboť pro daný kódový poměr R existuje minimální odstup šumu SNR, který je nutno dodržet a nazývá se *Shannonův limit*[2][3].

Dalším důležitým parametrem je *kódový zisk*. Protichybový kód tím, že opravuje chyby, snižuje bitovou chybovost BER při stávajícím SNR. Při snížení BER je možno akceptovat vyšší úroveň bílého šumu při zachování stejné informační rychlosti, nebo jinými slovy snížit odstup SNR při zachování stejné chybovosti. Kódový zisk je tedy poměrem SNR bez kódování a SNR s účinkem kódování. Je dán vztahem:

$$\gamma = 10.log\left(\frac{SNR}{SNR_k}\right) \quad [dB] \tag{1.5}$$

1.1.3 Základní parametry protichybových kódů

Pro popis principu zabezpečení a z nich vyplývajících schopností protichybových kódů je nutné si uvést několik hlavních pojmů. Nejdůležitějším z nich je **Hammingova vzdálenost d**. Tento parametr kódu udává počet prvků, ve kterých se liší dvě kódová slova. Udává tedy míru rozdílnosti (rozlišitelnosti) dvou posloupností prvků.



Obr. 1.2: Hammingova vzdálenost d

Na obrázku 1.2je uveden příklad všech kódových kombinací kódu o délce n = 3. Jedná se o vektorový prostor o velikosti $2^n = 2^3 = 8$ prvků, ve dvojkové soustavě čísel 0 až 7. V celém oboru vektorů tedy lze nalézt různé Hammingovy vzdálenosti, od d = 1 až po d = 3. Na obrázku je vlastně uveden kód bez přidané nadbytečné informace ve formě redundantních prvků, tedy tento kód obsahuje všechny kombinace prvků.

Lze tedy konstatovat, že u libovolného kódu může Hammingova vzdálenost nabývat nejméně $d_{min} = 1$ a nejvíce pouze $d_{max} = n$. V dalším popisu zabezpečovacích schopností kódů je důležitá pouze **minimální Hammingova vzdálenost** $\boldsymbol{d_{min}}$, viz dále.

Dalším pojmem je **Hammingova váha** $\boldsymbol{w}$. Tento parametr udává počet nenulových prvků v kódovém slově, samozřejmě kromě slova obsahujícího jen nulové prvky (nulového slova) :

$\boldsymbol{f_1}$ = 0 1 1 0 , tento vektor obsahuje dva nenulové prvky, tedy $w = 2$
$\boldsymbol{f_2}$ = 0 1 0 0 , $w = 1$
$\boldsymbol{f_3}$ = 1 1 1 1 , $w = 4$

I zde je důležitá **minimální Hammingova váha** $\boldsymbol{w_{min}}$, která udává minimální počet nenulových prvků ve skupině všech kódových slov daného protichybového kódu. w_{min} tedy zajišťuje určitou rovnoměrnost nenulových prvků v celém kódu. Pro *lineární blokové kódy* musí být zajištěna podmínka :

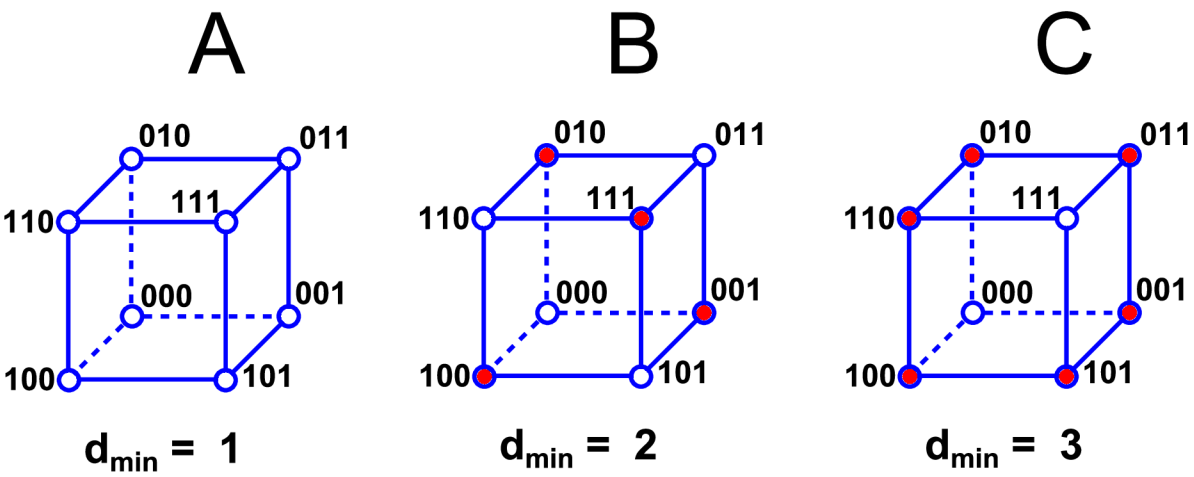
$$w_{min} = d_{min}$$

v celé množině kódových slov.

1.1.4 Schopnosti protichybových kódů

Zabezpečovacími schopnostmi kódů rozumíme detekční nebo opravné možnosti kódu, tedy kolik chyb je možné zjistit a kolik z nich opravit. Jak již bylo naznačeno výše v textu, tyto schopnosti lze zajistit zvyšováním hodnoty d_{min} v platných kódových slovech a tím zvyšování počtu neplatných kódových slov na úkor počtu platných kódových slov. Tímto zvyšujeme míru nadbytečnosti v kódu. Při přijetí neplatného slova přijímač (dekodér) **zdetekuje** a ohlásí chybu.

Při větším vektorovém podprostoru neplatných kombinací získáme i možnost **opravy** neboť vzniká situace, kdy je neplatné slovo nejvíce podobné právě jednomu platnému slovu. Tuto situaci lze velmi názorně demonstrovat na tzv. *Kódové krychli* [2] [3] [5]. Krychle se svými 8 rohovými body lze využít na vizualizaci kódu do kapacity $L = 2^3$. Vícerozměrná krychle není pro běžného člověka k demonstraci vhodná. Při bližším pohledu na obrázek 1.3 je snadno vidět, že hrana krychle odpovídá Hammingově vzdálenosti d = 1:



Obr. 1.3: Kódová krychle : bílé body jsou platné kombinace, červené neplatné

Z hlediska principu zabezpečení protichybovými kódy mohou nastat 3 triviální možnosti :

1. $d_{min} = 1$ (varianta krychle A)
   U této varianty využíváme všech kódových slov, tedy kapacita kódu $L = 2^n$. **Nelze ani detekovat ani opravit chybu.** Zde se nejedná o protichybový kód.
2. $d_{min} = 2$ (varianta krychle B)
   U této varianty využíváme jen kapacity kódu $L = 2^{n-1}$. Tento kód je schopen

detekovat 1 chybu, neboť je zde k dispozici právě jedna neplatná kombinace k jedné platné.

3. $d_{min} = 3$ (varianta krychle C)
   U této varianty využíváme kapacity kódu pouze $L = 2^n - 2$. Tento kód je schopen **detekovat 2 chyby**, neboť na jednu platnou kombinaci připadají 2 neplatné a **opravit 1 chybu**, neboť má tato kombinace od své platné kombinace nejmenší Hammingovu vzdálenost.

Na základě této analýzy je možné odvodit vztahy [2][3][5] pro výpočet zabezpečovacích schopností kódů. Pro počet opravitelných chyb se volí název t_1 a pro počet detekovatelných chyb t_2:

Detekční schopnost :

$$d_{min} = t_2 + 1 \quad \Longrightarrow \quad t_2 = d_{min} - 1 \tag{1.6}$$

Korekční schopnost :

$$d_{min} = 2t_1 + 1 \quad \Longrightarrow \quad t_1 = (d_{min} - 1)/2 \tag{1.7}$$

Smíšená schopnost :

$$d_{min} = t_1 + t_2 + 1 \tag{1.8}$$

Při návrhu protichybového kódu, kromě požadovaného kódového poměru a zisku, je nutno také respektovat jisté minimální hodnoty délky slova n a nadbytečnosti r, tak aby bylo dosaženo požadovaných hodnot t_1 a t_2. Tyto parametry určuje tzv. **Plotkinova hranice** a je dána těmito vztahy:

$$n \geq \frac{F.d_{min} - 1}{F - 1} \tag{1.9}$$

$$r \geq \frac{F.d_{min} - 1}{F - 1} - (1 + log_F.d_{min}) \tag{1.10}$$

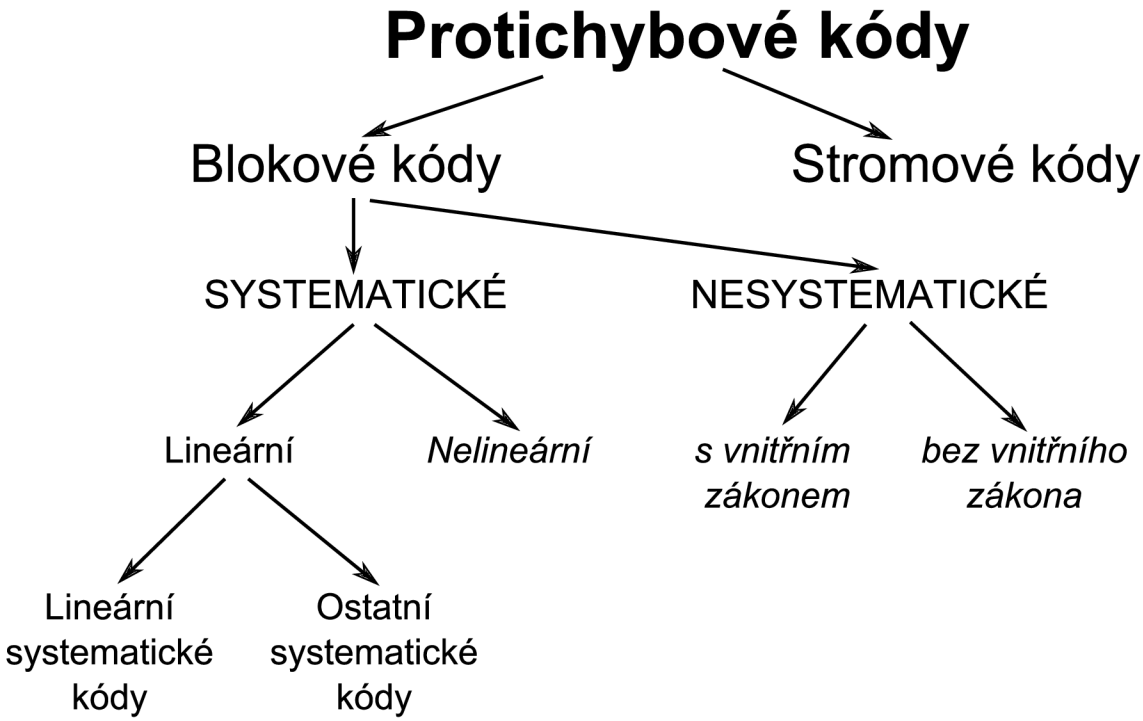
pro použití v binární soustavě, kde počet stavů signálu F = 2, lze vztahy zjednodušit takto :

$$n \geq 2.d_{min} - 1 \tag{1.11}$$

$$r \geq \log_2 . \frac{2^{2(d_{min}-1)}}{d_{min}} \tag{1.12}$$

1.1.5 Rozdělení protichybových kódů

Podrobnější třídění protichybových kódů se realizuje podle způsobu uspořádání redundantních zabezpečovacích prvků a lze v této podobě nalézt v různých informačních zdrojích [2][3][4][5] , viz obrázek 1.4.



Obr. 1.4: Třídění protichybových kódů

Tato práce se zabývá pouze kódy lineárními systematickými, proto v dalším textu bude uveden jejich popis.

1.2 Lineární blokové systematické kódy

Lineární blokové systematické kódy jak vyplývá z názvu patří mezi blokové kódy, které vyžadují dělení proudu informačních prvků na úseky pevné délky n a jako všechny systematické kódy ukládají zabezpečovací prvky r vždy do stejného definovaného místa, a to buď na začátek (v prostředí MATLAB) nebo na konec. V dalším textu této práce se bude vyskytovat pouze varianta druhá, tedy na konci kódového slova. Základní definicí těchto kódů je, že **lineární kód je podprostorem vektorového prostoru (podmnožinou všech kódových kombinací) a lze ho vytvořit několika generujícími vektory tohoto podprostoru za použití lineární algebry.**

Blokové kódy systematické lze dále rozdělit dle způsobu kódování na :

- Lineární blokové obecné - vytvářejí se pomocí ***vytvářecí matice G***
- Lineární blokové cyklické - vytvářejí se pomocí ***vytvářecího polynomu G(x)***

1.2.1 Vytvářecí matice lineárního blokového kódu

Při procesu zabezpečení lineárním kódem na vstupu kodéru získáváme blok posloupnosti informačních dat, který označujeme zprávou dané délky k a je ve tvaru vektoru:

$$\boldsymbol{p} = [p_1, p_2, p_3 ... p_k]$$

Kodér následně přidává zabezpečovací prvky v délce r. Tímto vznikne výsledné kódové slovo :

$$\boldsymbol{f} = [f_1, f_2, f_3 ... f_n]$$

které se odesílá přenosovou cestou k přijímači. Kódová slova **f** jsou vytvářena dle předpisu daného vytvářecí maticí **G**. Tato matice je složena jak již bylo uvedeno z vybraných kódových slov, z kterých lze lineární kombinací vytvořit každý vektor **f**, tedy celý kód. Vytvářecí matice **G** se nazývá báze vektorového podprostoru a je složena z generujících vektorů **g**.

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{g_1} \\ \boldsymbol{g_2} \\ \vdots \\ \boldsymbol{g_k} \end{bmatrix} = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix}$$

Princip kódování zprávy vychází z koeficientů lineárních kombinací p, které jsou přímo informačními prvky. Matice **G** je sestavena tak, aby prvek p_1 ovlivnil existenci generujícího vektoru $\boldsymbol{g_1}$ ve výsledné lineární kombinaci a tak dále. Matematicky lze tedy proces kódování zapsat jako násobení vektoru zprávy a vytvářecí matice :

$$\boldsymbol{f} = \boldsymbol{p} \,.\, \boldsymbol{G} \tag{1.13}$$

Generující vektor $\boldsymbol{g}$ je složen z prvků p a prvků r, lze matici **G** tedy zapsat v kanonickém tvaru :

$$\boldsymbol{G} = [\ \boldsymbol{I_{k \times k}} \quad \boldsymbol{C_{k \times r}}\ ] \tag{1.14}$$

kde $\boldsymbol{I_{k\times k}}$ je jednotková informační matice a $\boldsymbol{C_{k\times r}}$ je zabezpečovací matice. Příkladem vytvářecí matice je :

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

Při sestavování vytvářecí matice je nezbytné dodržet tyto zásady :

- Rozměr matice **G** musí být n × r
- Generující vektory **g** nesmí být lineárně závislé
- Nesmí obsahovat nulový vektor (kódové slovo s pouze nulovými prvky)
- Minimální Hammingova vzdálenost mezi řádky matice **G** musí být zároveň i minimální Hammingova vzdálenost celého kódu a zároveň musí být rovna w_{min}.

1.2.2 Kontrolní matice lineárního blokového kódu

Pomocí vytvářecí matice jsou data zakódována a tím je získáno kódové slovo, které je vysláno přes telekomunikační kanál. Na straně příjemce v dekodéru je potřeba zkontrolovat bezchybnost přenosu a právě za tímto účelem se používá tzv. kontrolní matice **H**. Tato matice obsahuje r řádků a n sloupců. Pro dekódovací proces se ale využívá transponované formě. Vycházíme ze základního vztahu :

$$\boldsymbol{G} \,.\, \boldsymbol{H^T} = \boldsymbol{0} \tag{1.15}$$

Matici **H** lze zapsat také v kanonickém tvaru, která umožňuje snadné vytvoření z matice **G** :

$$\boldsymbol{H^T} = \begin{bmatrix} \boldsymbol{C_{k\times r}} \\ \boldsymbol{I_{r\times r}} \end{bmatrix} \tag{1.16}$$

1.2.3 Detekce chyb

Vysílač po výstupu z kodéru posílá připravené zdrojové kódové slovo **f** přes komunikační kanál. Při přenosu dojde k ovlivnění původního vektoru chybovým vektorem **e**. Jak již bylo uvedeno, tento proces modelujeme pomocí vztahu :

$$\boldsymbol{j} = \boldsymbol{f} \oplus \boldsymbol{e}$$

Detekci chyb, v rámci schopnosti daného kódu, určíme snadno vynásobením kódového slova **j** s transponovanou kontrolní maticí. Výsledkem je stavový vektor **s**,

označovaný jako **syndrom** :

$$\boldsymbol{s} = \boldsymbol{j} \ . \ \boldsymbol{H^T} \tag{1.17}$$

Je-li tento syndrom roven nule, pak vektor **j** je roven původnímu vektoru **f**. Tento výrok platí jen při nepřekročení detekčních schopností kódu. Pokud je syndrom nenulový, k chybě došlo vždy.

1.2.4 Oprava chyb

V případě, že k dekódování byl použit kód s opravnými schopnostmi, je možné realizovat proces obnovení původní informace. Není nutné připomínat, že počet chyb, které lze opravit musí být menší nebo roven t_1. V opačném případě dostáváme syndrom stejný jako při opravě opravitelné chyby, ale výsledné opravené kódové slovo se nerovná originální zprávě **f**.

Proces opravy přijatého kódového slova lze popsat následujícím vztahem, jedná se o opačný postup při vzniku chyby :

$$\boldsymbol{f} = \boldsymbol{j} \ \oplus \ \boldsymbol{e} \tag{1.18}$$

Tedy hlavní podstatou opravného procesu je získat původní obraz chybového vektoru **e**. Následujícím vztahem lze ale dokázat, že jakýkoliv přijatý vektor **j** pozměněný stejným vektorem **e** poskytuje po vynásobení s $\boldsymbol{H^T}$ vždy stejný syndrom **s** [2][5] :

$$\boldsymbol{s} = \boldsymbol{j} \ . \ \boldsymbol{H^T} = (\boldsymbol{f} \oplus \boldsymbol{e}) \ . \ \boldsymbol{H^T} = \boldsymbol{f}.\boldsymbol{H^T} \oplus \boldsymbol{e}.\boldsymbol{H^T} = \boldsymbol{e}.\boldsymbol{H^T} \tag{1.19}$$

neboť $\boldsymbol{G}.\boldsymbol{H^T} = 0$ a tedy $\boldsymbol{f}.\boldsymbol{H^T} = 0$. Syndrom **s** tedy je součtem řádků $\boldsymbol{H^T}$ korespondujícím jedničkám v chybovém vektoru **e**, při 1 chybě jsou syndromy přímo řádky kontrolní matice.

V praxi je výpočet přiřazení $\boldsymbol{s} \longmapsto \boldsymbol{e}$ pro dekodér neoptimální a zbytečně zatěžuje procesor, proto se dekódování provádí vyhledáním vektoru **e** v předem připravené **Tabulce syndromů**. Tabulka obsahuje všechny kombinace chybového vektoru **e** a k nim jsou dopočítány syndromy **s**.

V dekódovacím procesu při vyhledávání syndromu je nutné procházet nejprve vektory **e** s nejmenší *Hammingovou váhou w*, jinak by mohlo dojít k překročení opravných schopností kódu. Jinou možností je zredukovat tabulku syndromů pouze na opravitelnou množinu chybových vektorů **e**.

1.3 Cyklické systematické kódy

Jak již bylo uvedeno, cyklické kódy jsou speciálním druhem lineárních blokových kódů. Jedná se o efektivní realizaci lineárního blokového kódu.[5] Cyklický kód je definován vytvářecím polynomem, na rozdíl od obecného lineárního kódu, který je definován vytvářecí maticí. Využití polynomu je velmi efektivní především pro konstrukci kodéru i dekodéru pomocí zpožďovacích článků a sčítaček.

1.3.1 Princip zabezpečení cyklickým kódem

Stejně jako u lineárního kódu, principem zabezpečení je přidání nadbytečné informace ke zdrojové zprávě. Cyklický kód je také určen vektorovým podprostorem, tedy má i svou vytvářecí a kontrolní matici, ale tuto vytvářecí matici lze převést lineární kombinací řádků do konvolučního tvaru a vyjádřit ji pouze polynomem $G(x)$. Všechny výpočty zabezpečovacího procesu realizujeme pouze s mnohočleny.

Přehled použitých mnohočlenů:

- $P(x)$ - mnohočlen bloku nezabezpečené zprávy
- $G(x)$ - vytvářecí mnohočlen
- $M(x)$ - mnohočlen podílu
- $R(x)$ - mnohočlen zbytku po dělení
- $F(x)$ - mnohočlen zabezpečené zprávy - kódového slova
- $J(x)$ - mnohočlen přenesené zprávy
- $S(x)$ - mnohočlen syndromu

Vlastní princip zabezpečení vyjadřujeme následujícím vztahem :

$$F(x) = P(x) \cdot x^{n-k} + R(x) = M(x) \cdot G(x), \tag{1.20}$$

tedy ke zdrojové zprávě přidáme takový polynom $R(x)$, aby výsledné dělení při dekódování bylo bezezbytkové. Tento polynom získáme ze vtahu :

$$P(x) \cdot x^{n-k} : G(x) = M(x) + R(x) \tag{1.21}$$

Násobením polynomem x^{n-k} posuneme posloupnost zdrojové zprávy o n-k doleva a získáme tím vhodný prostor pro přenos zabezpečovací informace R(x) v kódovém slově $F(x)$.

1.3.2 Princip detekce chyby při zabezpečení cyklickým kódem

Detekční schopnost cyklického kódu je dána stejným vztahem pro t_2 jako u lineárního kódu. Samotnou detekci provádíme výpočtem syndromu $S(x)$, který je nulový v případě bezchybného přenosu. V opačném případě je detekována chyba. Zde je opět vhodné připomenout, že i v případě nulového syndromu může dojít k chybě při přenosu, pokud bude chyb více než jsou detekční schopnosti kódu. V takovém případě přebírá detekci chyby vyšší přenosová vrstva. Tento stav snižujeme metodami jako je prokládání (Interleaving).

Syndrom získáme z následujícího vztahu :

$$S(x) = J(x) : G(x) \tag{1.22}$$

1.3.3 Princip korekce chyby při zabezpečení cyklickým kódem

Princip korekce kódového slova $J(x)$ je také obdobný jako u lineárního kódu. Z výše uvedeného vztahu získáme syndrom $S(x)$. Porovnáním v tabulce syndromů pak získáme chybový (nebo opravný) vektor $E(x)$, který přičteme k přijatému kódovému slovu :

$$F(x) = J(x) + E(x) \tag{1.23}$$

Posledním krokem je získání přenesené zprávy odstraněním n-k dlouhé posloupnosti zprava :

$$P(x) = F(x) : x^{n-k} \tag{1.24}$$

2 VÝUKOVÁ APLIKACE

Tato kapitola se zabývá popisem vývoje výukové aplikace, původními koncepčními záměry a stručně popisuje její realizaci. V poslední části kapitoly jsou diskutovány možnosti dalšího vývoje, jak technologicky tak i z hlediska obsahu aplikace. Jak již bylo uvedeno v zadání této práce, aplikace je realizována v prostředí MATLAB s využitím grafického rozhraní GUI.

2.1 Vývojové prostředí MATLAB

MATLAB (MATrix LABoratory) od firmy The MathWorks, Inc. bývá označován za světový standard pro technické výpočty. Obsahuje silné matematicko-grafické prostředí, v němž lze realizovat množství matematických výpočtů, modelování či analýzu fyzikálních dějů, ale i měření, analýzu a vizualizaci dat. Obecněji řečeno, v MATLABu lze provádět všechno, co lze matematicky popsat a výsledky i velmi efektně graficky vizualizovat [1].

MATLAB nabízí práci v základním prostředí, nebo je možné dokoupit rozšiřující moduly nazývané Toolboxy. Pro návrh komunikačních zařízení je k dispozici Toolbox Communications, kterým se ale tato práce nezabývá. Systém MATLAB byl v prvních verzích původně navržen pro operace lineární algebry a v dnešním prostředí tohoto systému je tato koncepce stále přítomna. MATLAB považuje všechny proměnné za matice (s různými rozměry) a prvky matic mohou být nejen čísla, ale i další prvky včetně obrázků. Pro grafické prostředí je k dispozici GUI se strukturou jednotlivých grafických objektů typu *figure*, které poskytuje základní uživatelské rozhraní, ale i možnosti 3D grafiky. Pro snadné a rychlé návrhy GUI je k dispozici editor Guide, který generuje příslušný kód.

2.2 Základní koncepce

Aplikace jak již bylo uvedeno v zadání, je vyvinuta pro prostředí MATLAB. Program se v současné finální verzi skládá z hlavního menu, jako oddělené funkce *m-file* **Main.m** s 80 řádky, dále z algoritmu demonstrace blokových kódů **Blokove_kody.m** s 1590 řádky a z algoritmu demonstrace cyklických kódů **Cyklicke.m** s 1091 řádky.

Pro každou část aplikace byl zvolen odlišný koncept řešení, pro demonstraci různých možností přístupu programování v prostředí MATLAB. Pro část cyklických kódů byl výrazně změněn návrh designu, výběr kódu s určením jeho schopností, výběry binárních slov z předdefinované nabídky a plné využití funkcí z rozšíření

Communications System Toolbox. V sekci lineárních kódů jsou naopak všechny algoritmy navrženy vlastní. Hlavním zlepšením v sekci cyklických kódů je beze sporu zobrazení všech textových prvků využitím prvků **axes** a dostatek specifických funkcí, především funkce pro zobrazení polynomu a matice s možností přednastavení minimálního a maximálního fontu. U velkých matic se automaticky přidají slidery pro posun matice v definovaném okně pro **axes**.

Základní koncepcí lineárních blokových kódů při psaní zdrojového kódu byla zvolena metoda **Switched Board Programming**. Jedná se tedy o pouze jednu funkci uloženou v jednom souboru *m-file*, při které je kód v hlavní části této funkce. Všechny další funkce se realizují voláním původní funkce s různými vstupními parametry (*varargin*). Jedním z parametrů při opětovném volání hlavního programu je návěstí, kterým se podle řídicího příkazu **Switch** určuje, který podprogram se spustí. Určitou nevýhodou této metody může být nárůst počtu řádků zdroje při větším projektu, výhodou je přehledné řazení podfunkcí do jednoho bloku. Zjednodušená ukázka algoritmu pro krok č.1 je na obrázku 2.1. **Callback** je zpětná vazba na vstup uživatele, realizující postup v algoritmu.

Samotná vizualizace je realizována dle zadání uživatelským grafickým rozhraním GUI. Skládá se z jednoho objektu *Figure*, který si své výchozí rozměry přebírá z nadřazeného objektu Root, což představuje celou obrazovku s jejím použitým rozlišením. Využívá se k tomu globální proměnné **Monitor** a tyto hodnoty se aktuálně přepočítávají pro rozměr hlavního okna aplikace. Dále rozhraní využívá podřízených objektů typu *uicontrol* pro ovládání akcí, zobrazení textu a editačních polí. Tyto objekty jsou podřízeny hlavnímu oknu a využívají normalizovaných jednotek vztažených k hlavnímu oknu *Figure*, takže při změně rozlišení monitoru se přepočítávají zcela automaticky.

Pro zobrazení vektorů a matic byl zvolen objekt *uitable*. Byl zvolen pro snadné vypsání i delších matic, se kterými program počítá. Delší kódy je nutno samozřejmě rolovat posuvníky, které objekt *uitable* automaticky přidává při překročení grafických možností zobrazení.

Pro vizualizaci jednotlivých kroků zabezpečení byly zvoleny oddělené objekty uicontrol pro jejich možnost nezávislé animace. Program je připraven pro animaci zpožďováním zobrazení prvků, tedy postupným zobrazením např. jednotlivých prvků matematického vzorce. Rychlost animace lze parametrizovat proměnnou *a*. Tato funkce není v aktuální verzi použita.

Využití proměnných je řešeno globálními proměnnými vyskytujícími se v celém programu a jejich názvy jsou voleny co nejblíže jejich matematickému vyjádření. Adresování objektů GUI pomocí jejich proměnných zvaných **Handles** bylo řešeno jejich globalizací a pro optimalizaci počtu těchto proměnných byla použita struktura **B**, která zahrnuje svoje podřízené objekty, například : tabulka matice **G** je

adresována pro získání jejích parametrů takto : *get(**B.MaticeGtable**)*.

Přehled nejdůležitějších proměnných použitých v programu :

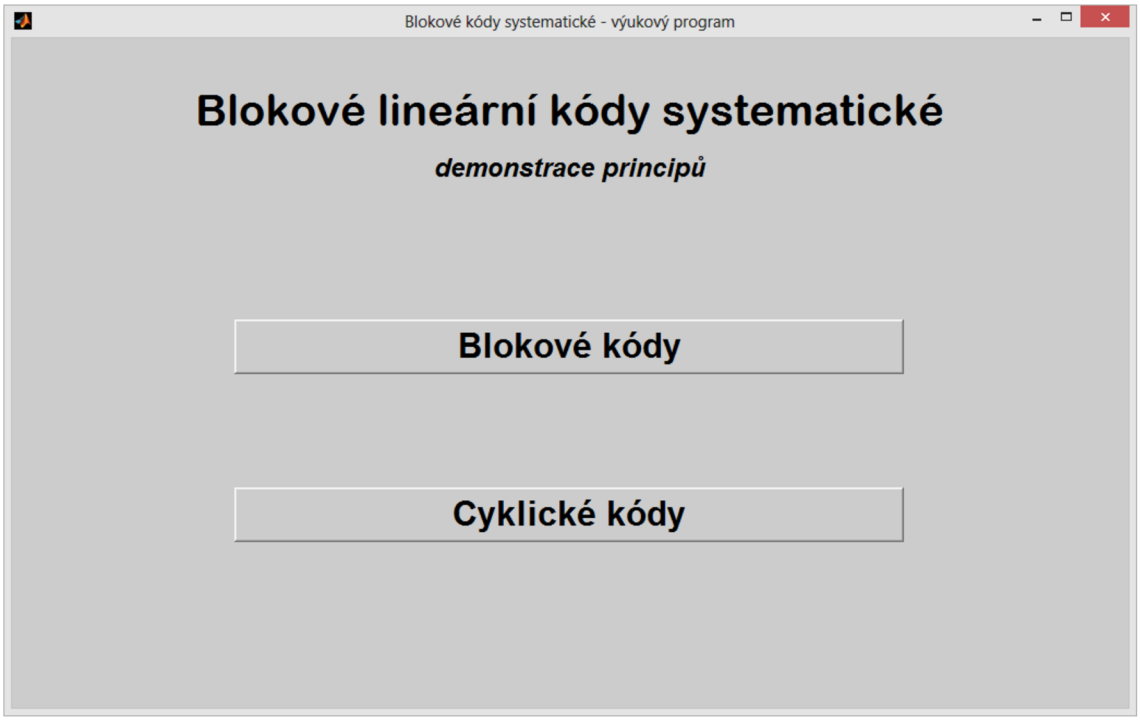
- global **Monitor** - získání rozlišení obrazovky
- global **B** - struktura obsahující všechny grafické prvky
- global **Kod** - matice obsahující platné kódové kombinace
- global **n** - délka kódového slova (codeword)
- global **k** - počet informačních prvků
- global **r** - počet zabezpečovacích prvků
- global **t1** - opravná schopnost kódu
- global **t2** - detekční schopnost kódu
- global **G** - generující matice
- global **Ht** - transponovaná kontrolní matice H
- global **p** - zdrojová zpráva
- global **f** - bezchybné kódové slovo na výstupu z kodéru
- global **j** - modifikované kódové slovo na vstupu dekodéru
- global **e** - chybový vektor
- global **s** - vektor syndromu získaného při procesu dekódování
- global **ff** - opravený vektor - musí se rovnat původnímu slovu f
- global **ee** - opravný vektor - vektor, kterým realizujeme opravu slova
- global **pp** - opravená zpráva ke kontrole s originální zprávou p
- global **a** - Časové zpoždění při animaci prvků

Koncepce celkového designu aplikace byla zvolena ve formě jednotlivých kroků algoritmu, připomínající prezentaci pomocí slajdů. Základní okno aplikace využívá téměř celé plochy monitoru, pro snadné doplňování prvků i názorné zobrazení a zdůraznění jednotlivých principů. Základní okno je rozděleno do 3 vyhrazených ploch: Hlavička prezentace, hlavní panel pro vizualizaci kde se zobrazuje celý proces a stavový panel s aktuálním stavem proměnných. Tento panel je záměrně dlouhý přes celou délku základního okna, aby bylo možno zobrazit výsledky zabezpečení až 64-bitových kódových slov. Stavové proměnné jsou zobrazovány pod sebou tak, aby bylo možno snadno porovnávat bezchybnost opravného procesu (lze porovnat zprávu **p** s výslednou **p'** a podobně). Dále v každém kroku je zobrazována stručná teorie týkající se tohoto kroku zabezpečení. Je psána velmi stručně, ale zároveň tak, aby vystihovala podstatu procesu a poukazuje na důležitá fakta. Jsou to spíše poznámky lektora, než kompletní vysvětlení teorie, proto se předpokládá studentovo načtení teorie ze skript nebo z teoretické části této práce 1.

Obr. 2.1: Ukázka algoritmu GUI aplikace

2.3 Realizace aplikace

2.3.1 Základní menu aplikace

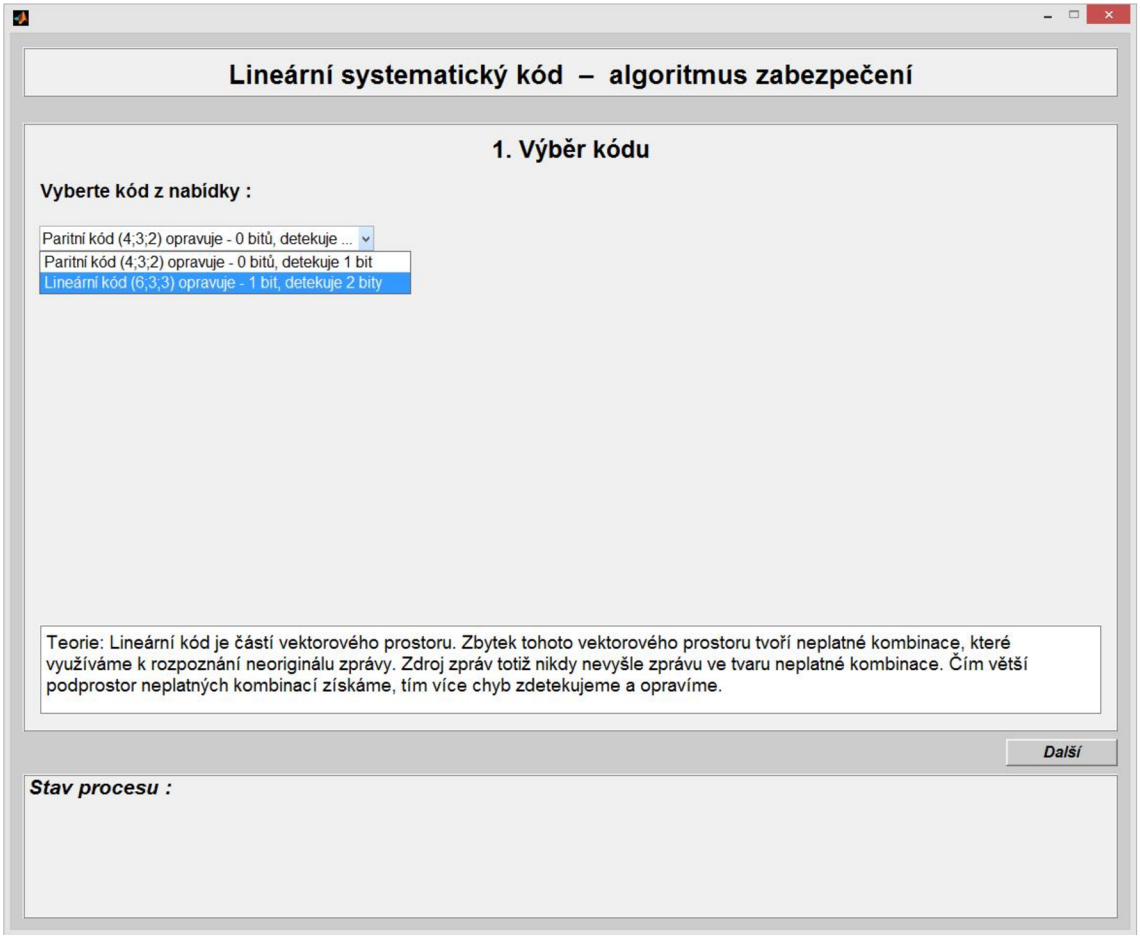


Obr. 2.2: Hlavní menu aplikace

Hlavní menu aplikace je řešeno jednoduchým stylem, se dvěmi tlačítky pro volbu konkrétního algoritmu. Je zde i tlačítko Cyklické, které je připraveno pro rozšíření o cyklické kódy. Menu kontroluje nastavení monitoru a při rozlišení menším než 1280x1024 varuje uživatele, že zobrazení nemusí být optimální. Všechny prvky GUI jsou sice parametrizované a zobrazit je lze i na nižších rozlišeních, ale zmenšení některých fontů je již neakceptovatelné.

Po stisku tlačítka *Blokové kódy* se spustí samostatná funkce **Blokove_kody.m**, která obsahuje kompletní řešení vizualizace blokových kódů. Po svém ukončení se znovu volá toto hlavní menu **Main.m**.

2.3.2 Výběr kódu



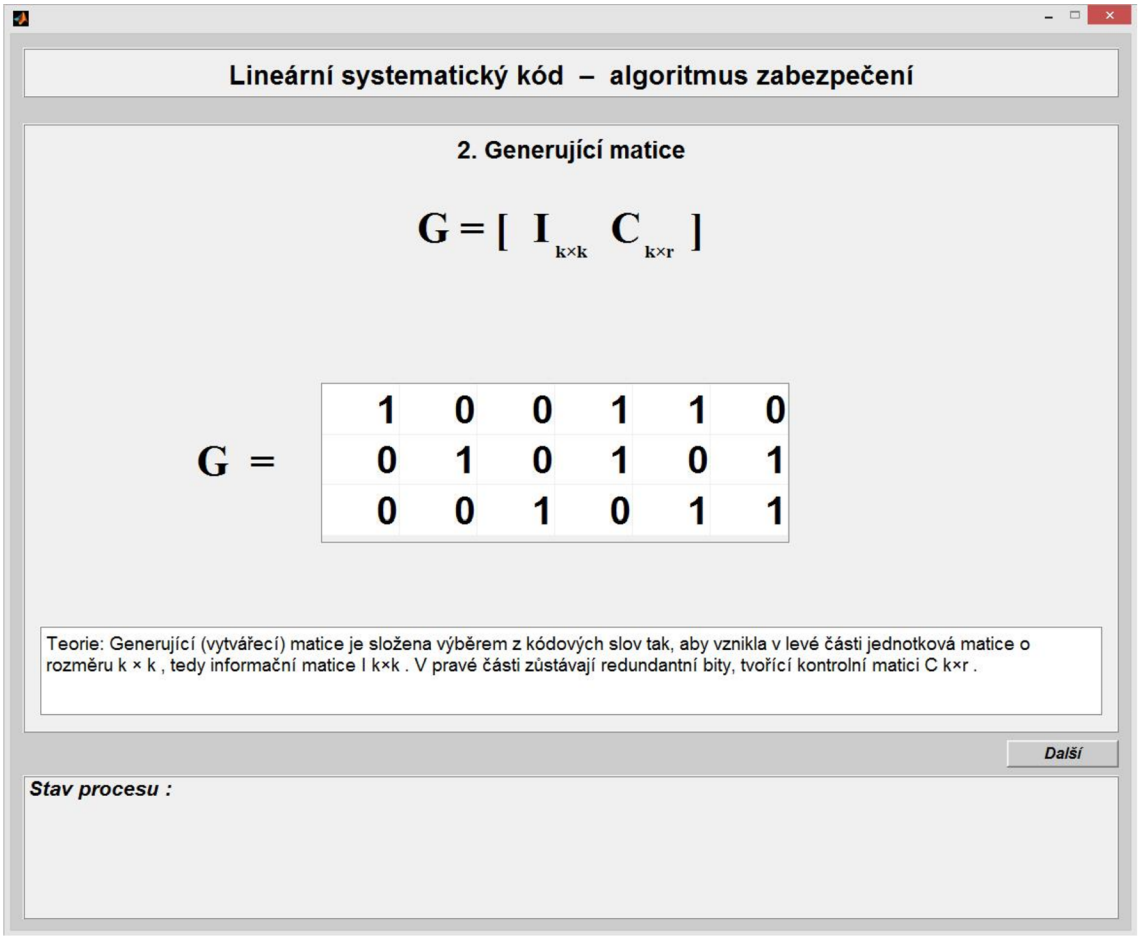
Obr. 2.3: Výběr kódu

Při spuštění hlavní funkce *Blokove_kody.m* bez parametru dojde k celkové inicializaci algoritmu, tedy k deklaraci všech globálních proměnných a k načtení všech prvků GUI, s ponechaným parametrem *Visibility* ve stavu *Off*.

V kroku výběru kódu lze vybrat kód pro další zpracování. Výběr je realizován prvkem *uicontrol* stylu *Popup-Menu*, kde jsou předdefinované lineární kódy s jejich případnými názvy a určením jejich schopností ve tvaru opravných t_1 a detekčních t_2. V aktuální verzi aplikace není možné zadat vlastní kód, který by aplikace zanalyzovala, potvrdila jeho použitelnost a vypočetla jeho schopnosti - tento složitější algoritmus je předmětem dalšího zpracování.

V tomto bodě podle výběru kódu dojde k naplnění všech potřebných proměnných, tedy n , k , r , $\mathbf{Kod}$, $\mathbf{G}$, $\boldsymbol{H}^{\boldsymbol{T}}$, t_1 a t_2. Proměnná $\mathbf{Kod}$ je maticí obsahující všechny platné varianty kódu, ze kterých lze vypočítat jeho parametry.

2.3.3 Generující matice



Obr. 2.4: Generující matice

V následujícím kroku je vizualizován princip vzniku generující matice **G**, vysvětlen v zobrazené nápovědě v panelu Teorie. Matice je zobrazena pomocí prvku uitable, takže i při větších rozměrech kódu lze rolovat a zobrazit celý kód.

Příklad kódu pro zobrazení tabulky s maticí **G** :

```
B.MaticeGtable = uitable('Data',G,...
                 'Enable','on',...
                 'Units','Pixels',...
                 'FontUnits','Pixels',...
                 'FontSize',TableFont,...
                 'FontWeight','bold',...
                 'position',[(320/1280)*Monitor(3)  400/1024*Monitor(4) (482/1280)*Monitor(3)
                      164/1024*Monitor(4)],...
                 'RowName',[],...
                 'ColumnName',[],...
                 'RowStriping','off',...
                 'ColumnFormat',{'short'},...
                 'ColumnWidth',{80/1024*Monitor(4)},...
                 'Visible','off');
```

2.3.4 Zadání zdrojové zprávy

Obr. 2.5: Zdrojová zpráva

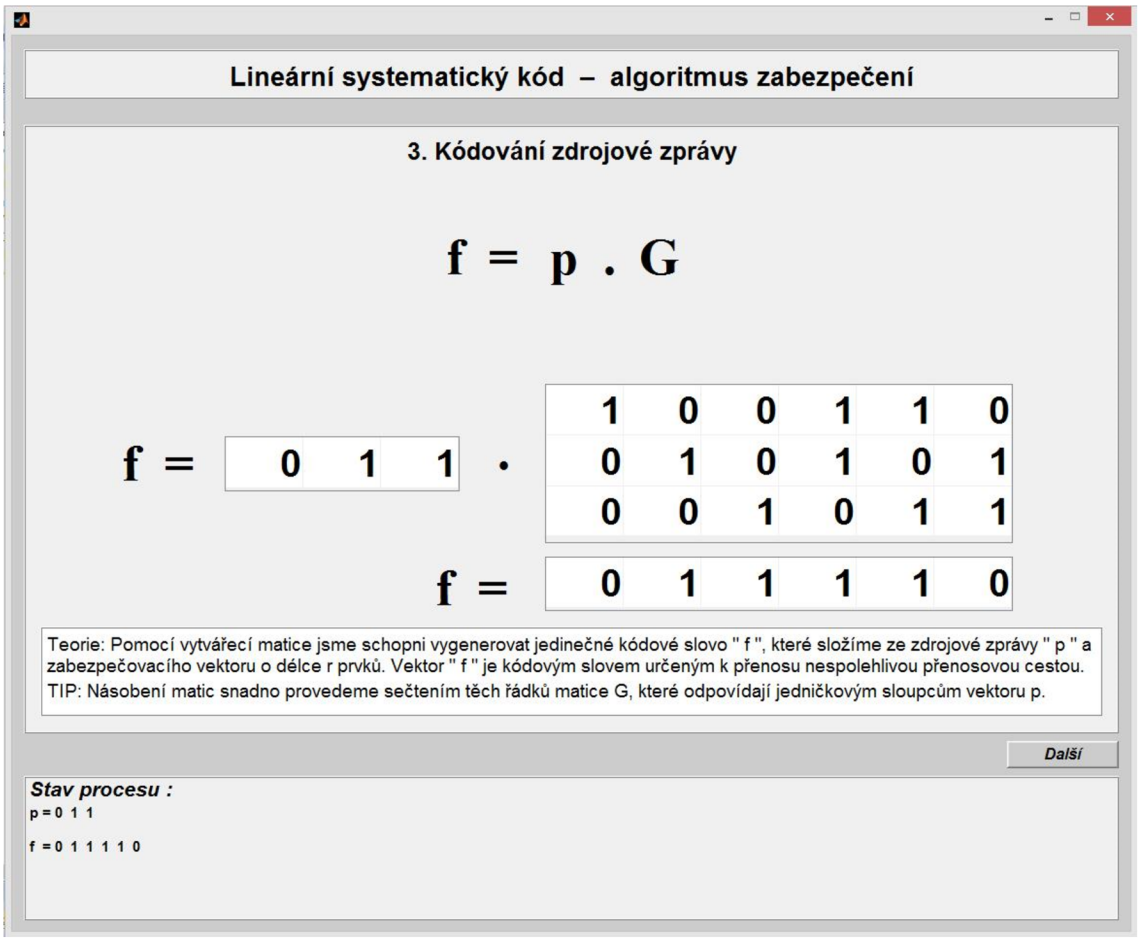
Zadání vstupní zabezpečované zprávy je provedeno jako samostatná akce v rámci jedné obrazovky. Z původního záměru sloučit do následujícího kroku kódování bylo upuštěno pro zachování prostoru pro následující krok.

Pro vstup zprávy je využit prvek *uicontrol* stylu *edit*. Vstup je možné naplnit použitím tlačítka Auto (stylu *Push button*), které vygeneruje binární číslo délky k a náhodné váhy w. Vstup je také ošetřen na vstup jen korektního čísla délky k.

Vstup zcela záměrně neomezuje zadání zdrojové zprávy s větší Hammingovou váhou než jsou detekční schopnosti kódu. Je to za účelem vyzkoušení možnosti překročení schopnosti kódu.

Nápověda v programu ve formě panelu Teorie je doplněna nápovědami formou parametru *ToolTipString* u některých prvků *uicontrol*, jak je zachyceno na obrázku2.5

2.3.5 Kódování zdrojové zprávy

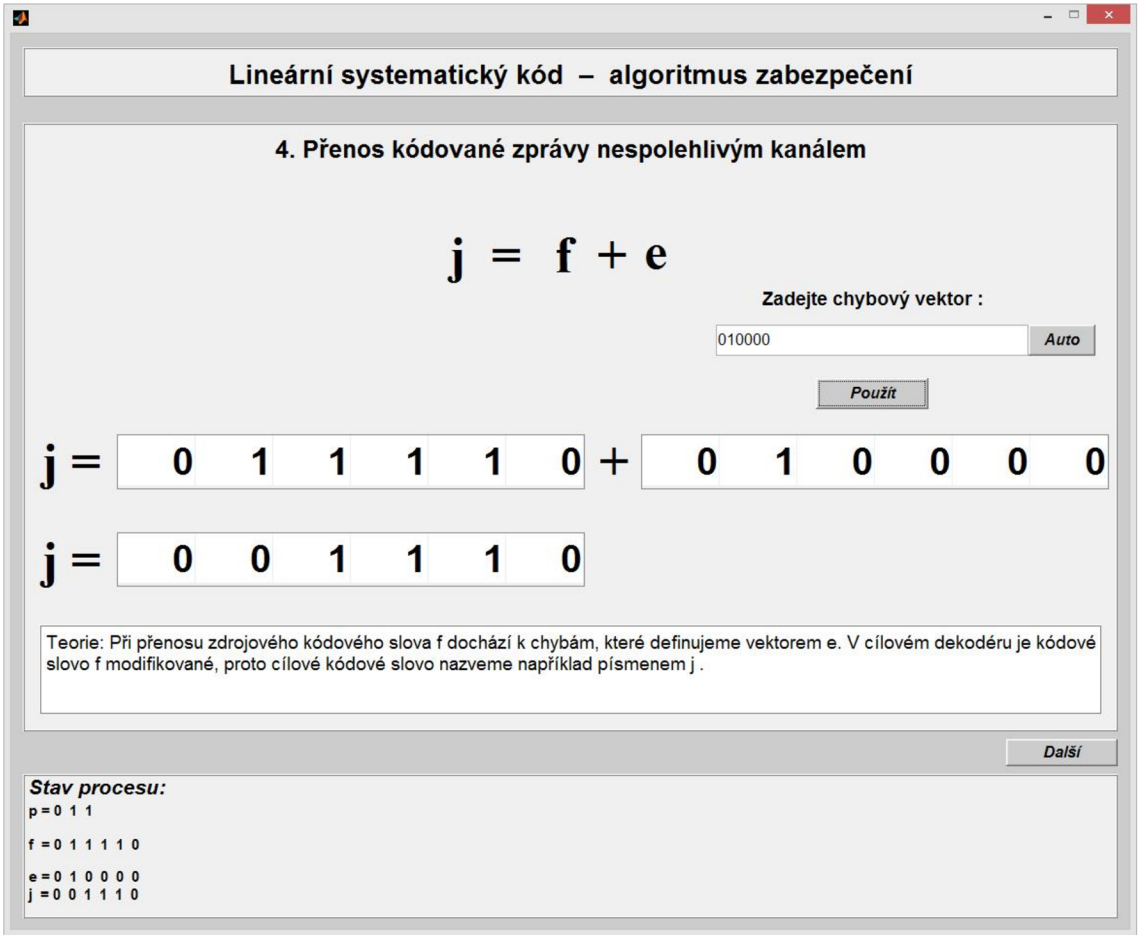


Obr. 2.6: Kódování

Procesu kódování je také věnována celá obrazovka. Je zde vizualizován vztah pro kódovací proces, a zobrazeny vektory ve formě *uitable* s dostatečně výraznou velikostí fontu. Výsledný vektor **f** je zarovnán pod matici **G** pro snadnou kontrolu jejich výpočtu.

V panelu *Stav procesu* již lze pozorovat stavy vektoru **p** a vektoru **f**.

2.3.6 Přenos kódované zprávy

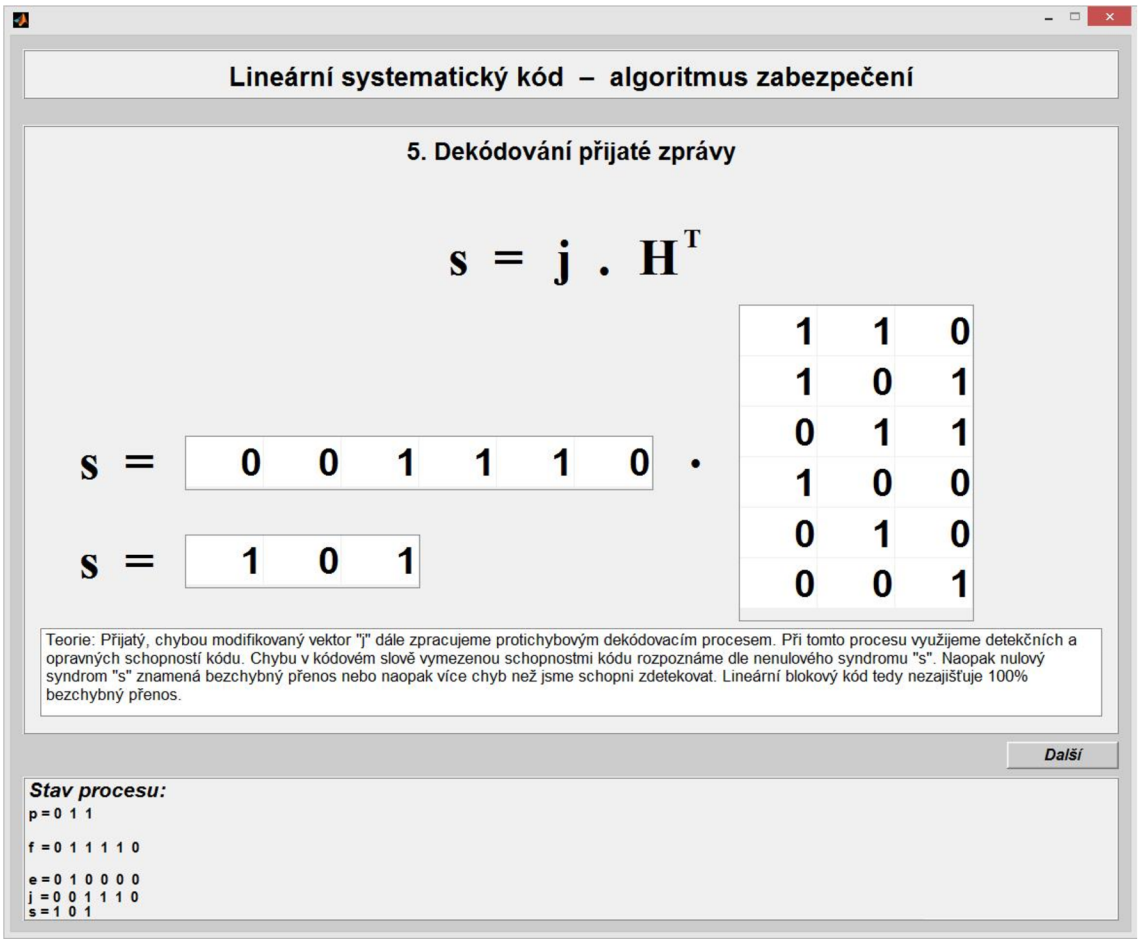


Obr. 2.7: Přenos kódované zprávy

V tomto kroku dochází k přenosu kódového slova **f**. Jeho ovlivnění chybovým vektorem **e** je vizualizováno opět objektem *uicontrol* stylu *edit* v rámci této obrazovky a je zarovnáno s tabulkou zobrazující vektor **e** pro názornost. Vstup je opět ošetřen na délku slova **n** a možnost naplnění náhodnou kombinací správné délky.

V této fázi algoritmu je kontrolována Hammingova váha w chybového vektoru a při překročení hodnoty t_1 dojde k varování o překročení opravných schopností kódu ve stavovém panelu. Při překročení hodnoty t_2 je zobrazeno analogicky varování o překročení detekčních schopností2.10. Algoritmus však pokračuje dále pro zobrazení dalších kroků.

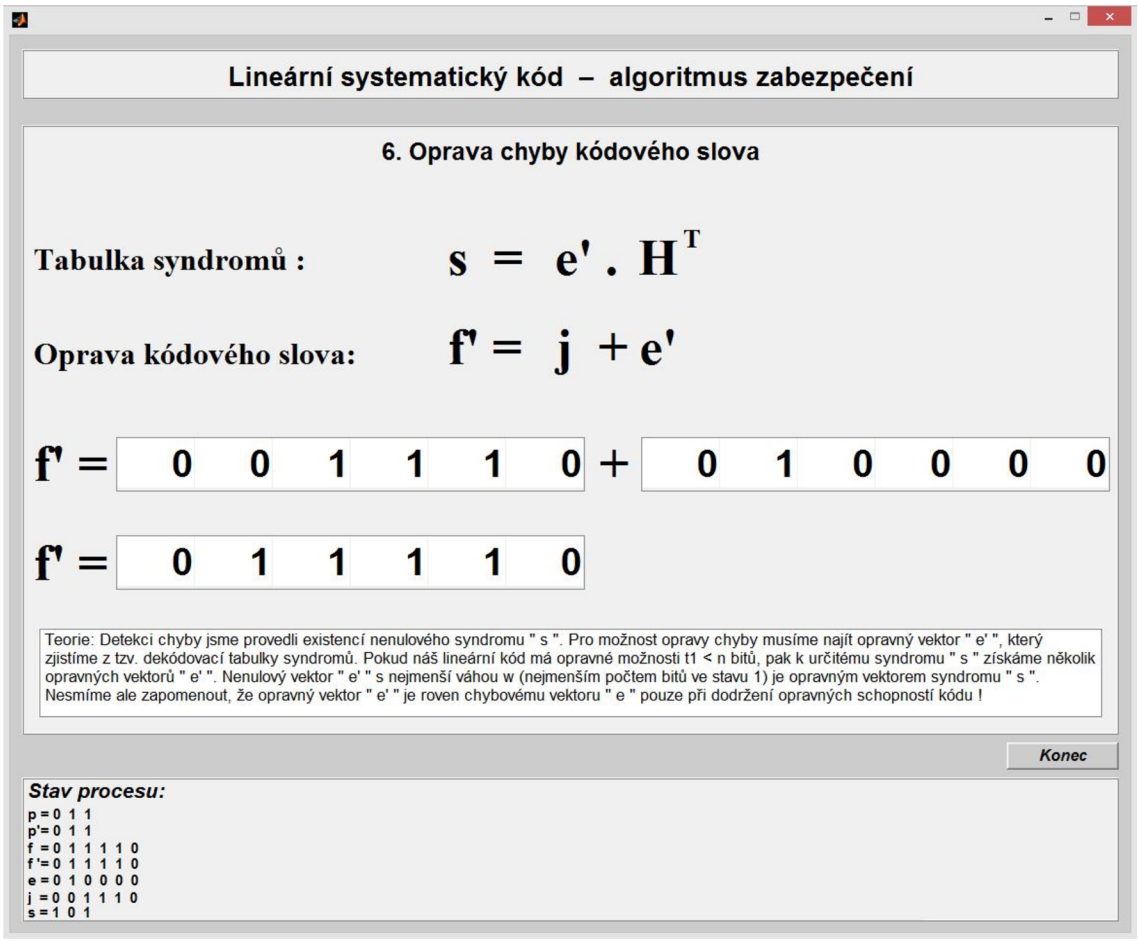
2.3.7 Dekódování přijaté zprávy



Obr. 2.8: Dekódování přijaté zprávy

V dekódovacím procesu je také proveden výpočet dle známého vztahu a získán syndrom **s**. Zde můžeme zkontrolovat kladný výsledek dekódovacího procesu existencí nulového syndromu, který je rovněž jako u předchozích kroků zobrazen v panelu statusů.

2.3.8 Oprava chyby kódového slova



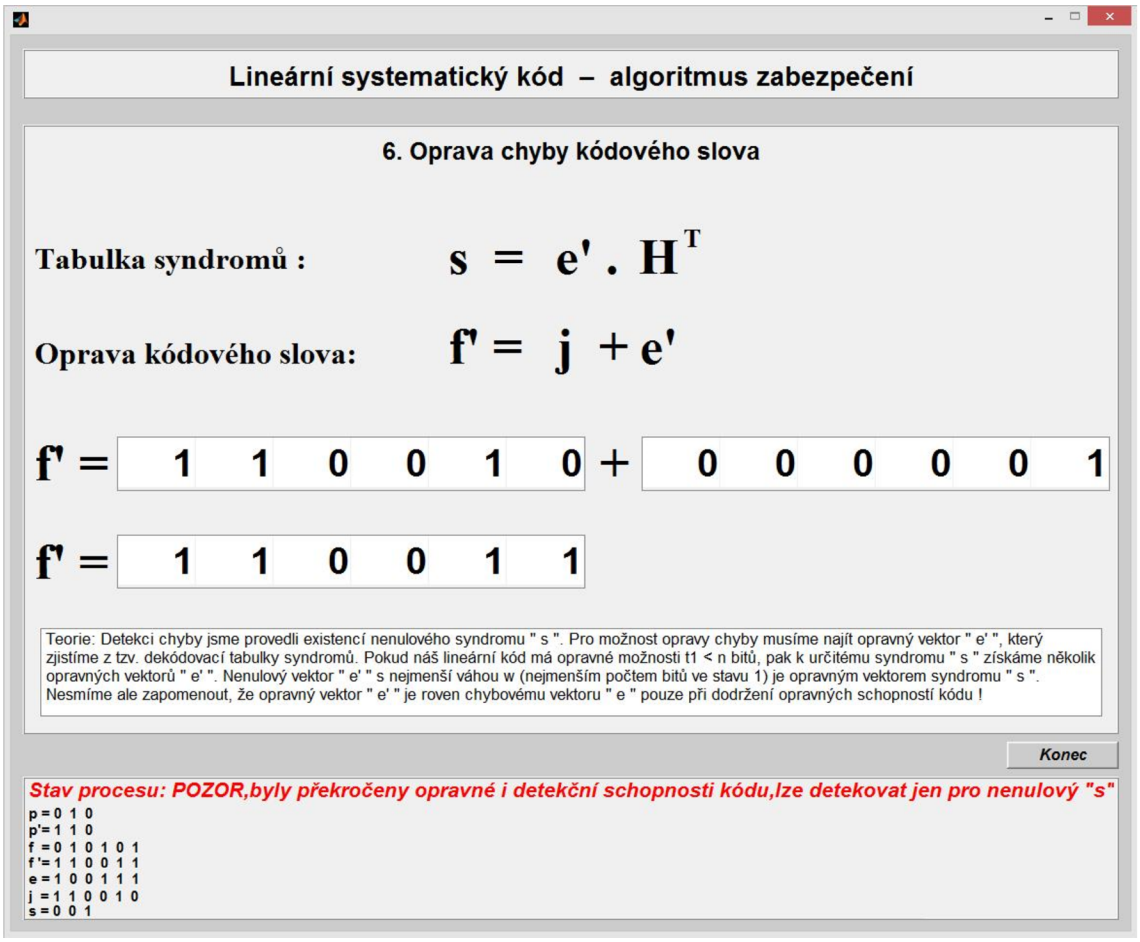
Obr. 2.9: Oprava chyby kódového slova

V tomto posledním kroku kódovacího procesu je využit nejsložitější algoritmus při vyhledávání tabulky syndromů. Pro zachování jednoduchosti a transparentnosti vizualizace není tabulka vytištěna celá, ale pouze vztah k jejímu výpočtu.

Přesněji řečeno, v tomto kroku výukového programu je vytvořen rozhodovací algoritmus, který generuje tabulku syndromů od vektorů **e** s nejmenším počtem chyb (nejmenší váhou w). Ke každému vektoru **e** dopočítá syndrom a tento ihned porovná se získaným syndromem **s**. V případě shody je určen chybový vektor s nejmenším možným počtem chyb a je použit v následné opravě přijatého slova **j**. Tímto postupem je zajištěno, že syndrom nebude přiřazen chybovému vektoru s více chybami než jsou v opravných možnostech kódu. Pro zrychlení dekódovacího procesu je vhodnější vygenerovat celou tabulku syndromů a uložit ve formě matice.

Opravené vektory **f** a **p** ovšem nejsou ty stejné proměnné, jako na vstupu při kódování, proto byly označeny čárkovanou variantou **f'** a **p'**. Tyto již lze snadno porovnat zobrazením ve stavovém panelu.

Tlačítkem Konec se ukončí příkazem *close all* všechny objekty GUI a spustí se hlavní menu **Main.m**.



Obr. 2.10: Oprava chyby kódového slova s překročením hodnot t_1 i t_2

2.4 Realizace aplikace - část cyklické kódy

Program Cyklické kódy byl výrazně změněn již při návrhu designu, struktuře skriptu i optimalizaci využití GUI objektů. Základní koncepce celoobrazovkové prezentace i principy vizualizace jsou podobné.

Obr. 2.11: Ukázka návrhu designu sekce Cyklické kódy

Mezi hlavní rozdíly od výše popsané realizace sekce blokových kódů patří:

- Výběr kódu s určením jeho schopností a generováním polynomu i matic
- Výběry binárních slov z předdefinované nabídky
- Plné využití funkcí z rozšíření Communications System Toolbox
- Zobrazení všech textových prvků využitím objektů **axes**
- Funkce pro zobrazení polynomu a matice s možností přednastavení minimálního a maximálního fontu. U velkých matic využití sliderů pro posun matice v definovaném okně pro **axes**.

2.5 Další vývoj aplikace

Aktuální stav výukové aplikace ve své verzi dává prostor pro další vývoj a vylepšení jak po technologické tak i obsahové stránce. Přestože část aplikace cyklické kódy je mnohem propracovanější než původní část lineární kódy, je vhodné aplikaci doplnit především o detailní vizualizace vlastních výpočtů (násobení matic, dělení polynomů a další).

Mezi hlavní body k vyšší efektivitě výukové aplikace patří :

- Kompletní přepracování designu sekce blokové kódy nebo přímé sloučení do jednoho objektu figure se sekcí cyklické kódy.
- Zadání chybového vektoru počtem chybných bitů, náhodně, případně kombinací všech metod
- Vizualizace dělení polynomů
- Vizualizace dělení polynomů postupným sčítáním
- Násobení matic
- Zobrazení celé tabulky syndromů s vyznačením aktuálního syndromu
- Tvorba vlastního blokového kódu zadáním parametrů a kontrolou Plotkinovou hranicí.

3 ZÁVĚR

Cílem této práce bylo vytvořit výukovou aplikaci, ve které je možné kromě teoretického základu si také ověřit a kontrolovat celý proces zabezpečení lineárními a cyklickými blokovými kódy.

V první části této práce je uveden teoretický základ pro efektivní práci s výukovým programem. Tato teorie je však rozvedena od základních principů vzniku chyby až po proces její korekce v dekodéru, neboť správné a kompletní pochopení principu i důvodu použití zabezpečení protichybovými kódy je základním cílem této práce.

Druhá část práce se zabývá popisem samotné realizace výukové aplikace. Krok po kroku je představen algoritmus programu s vizualizací jednotlivých kroků, ale i celková koncepce programu. Nakonec této části jsou nastíněny další směry a možnosti zdokonalení aplikace jak po stránce obsahové tak i technologické.

Poslední částí je samotný zdrojový kód výukové aplikace. Tato aplikace splňuje požadavky zadání jako výukové aplikace demonstrující principy protichybového zabezpečení blokovými a cyklickými kódy.

LITERATURA

[1] Zaplatílek, K., Doňar, B. *MATLAB: tvorba uživatelských aplikací.* Praha: BEN, 2004. ISBN 80-7300-133-0.

[2] Morelos-Zaragoza, R. *The art of error correcting coding.* Chichester: John Wiley & Sons, 2002. ISBN 0-471-49581-6.

[3] Lin, S., Costello, D. J. *Error Control Coding: Fundamentals and Applications* Prentice Hall: Englewood Cliffs, NJ, 2005 ISBN: 0-13-042672-5.

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

[5] Šilhavý, Pavel *Datová komunikace.*Vydání 1.Brno: Vysoké učení technické, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. ISBN:978-80-214-4455-3.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BER	Bit Error Rate - bitová chybovost
n_ch	počet chybných bitů - při měření bit po bitu
n_c	celkový počet bitů - při měření bit po bitu
ITU-T	ITU Telecommunication Standardization Sector is one of the three sectors of the International Telecommunication Union (ITU)
SNR	Signal to Noise Ratio - odstup výkonu signálu od výkonu šumu
SNR_k	Signal to Noise Ratio - odstup výkonu signálu od výkonu šumu kódovaného signálu
dB	decibel - obecné měřítko podílu dvou hodnot, logaritmická jednotka
XOR	exclusive OR - logický součet odpovídající operaci modulo 2
E(x)	polynomické vyjádření chybového slova
F(x)	polynomické vyjádření zdrojové zprávy
J(x)	polynomické vyjádření přijaté zprávy
R	kódový poměr R, code rate
n	délka kódového slova
k	délka zabezpečované zprávy - počet informačních prvků
r	počet zabezpečovacích prvků
d	Hammingova vzdálenost
d_{min}	minimální Hammingova vzdálenost
w	Hammingova vzdálenost
w_{min}	minimální Hammingova váha
t_1	počet opravitelných chyb
t_2	počet detekovatelných chyb
F	počet stavů signálu
G	vytvářecí matice lineárního kódu

$\mathbf{H}$	kontrolní matice lineárního kódu
$\boldsymbol{H^T}$	transponovaná matice lineárního kódu
$\mathbf{I}$	informační jednotková submatice
$\mathbf{C}$	zabezpečovací submatice
$\mathbf{e}$	vektor chybového slova
$\mathbf{f}$	vektor vyslaného kódového slova
$\mathbf{j}$	vektor přijatého kódového slova
$\mathbf{s}$	vektor syndromu dekódovacího procesu
MATLAB	MATrix LABoratory - vývojové prostředí od firmy The MathWorks, Inc.
GUI	Graphical User Interface - grafické uživatelské prostředí aplikace

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

Výpis souborů přiložených na CD :

- Main.m - Hlavní menu aplikace - testováno v Matlab verze 2013a
- Blokove kody.m - Sekce blokové kódy - testováno v Matlab verze 2013a
- Cyklicke.m - Sekce cyklické kódy - testováno v Matlab verze 2013a
- alg1.png - bitmapy pro zobrazení algoritmu 1-6
- alg2.png
- alg3.png
- alg4.png
- alg5.png
- alg6.png