# Kódování – cyklické kódy

Coding – cyclic code

Jakub Kettner

Bakalářská práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Jakub KETTNER** |
| Studijní program: | **B 3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Téma práce: | **Kódování -- cyklické kódy** |

Zásady pro vypracování:

1. **Seznamte se problematikou efektivních a bezpečnostních kódů, zaměřte se na kódy cyklické.**
2. **Zpracujte literární rešerši na dané téma.**
3. **Vypracujte názorné příklady.**
4. **Na základě teorie naprogramujte algoritmus kódování a dekódování s grafickou demonstrací funkcí.**
5. **Zhodnoťte dosažené výsledky.**

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:


1. **Zelinka, I., Prokopová Z.: Základy informatiky. FT UTB, Zlín, 2005, ISBN 80-7318-299-8.**
2. **Birkhoff, G., Bartee, T., C.: Aplikovaná algebra, Alfa, Bratislava, 1981.**
3. **Vlček, K.: Komprese a kódová zabezpečení v multimediálních komunikacích, Praha,BEN - technická literatura, 2004, ISBN 80-86056-68-6.**
4. **Kocurek, P., Novák, J.: Přenos informace. ČVUT, Praha, 2004, ISBN 8001028925.**
5. **Jiroušek, R.: Principy digitální komunikace. Leda, Voznice, 2006, ISBN 80-7335-084.**


Vedoucí bakalářské práce: **RNDr. Ing. Miloš Krčmář**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **20. února 2008**

Termín odevzdání bakalářské práce: **5. května 2008**

Ve Zlíně dne 20. února 2008

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
*děkan*

doc. Ing. Ivan Zelinka, Ph.D.
*ředitel ústavu*

## ABSTRAKT

Tato bakalářská práce poskytuje stručný přehled problematiky týkající se cyklických kódů. Teoretická část obsahuje krátký úvod do oblasti bezpečnostních lineárních kódů, základní matematickou teorii cyklických kódů a vysvětlení principů kódování a dekódování. Uvedena je též kapitola, kde je popsána základní konstrukce kodéru a dekodéru cyklického kódu. Praktická část obsahuje informace o programu pro simulaci cyklických kódů a o technologiích, které byly využity k jeho vytvoření.


Klíčová slova: cyklické kódy, kódování, dekódování, kodér, dekodér, konečná tělesa, Java


## ABSTRACT

This Bachelor thesis is offering a brief overview of a problematic concerning cyclic codes. The theoretical part is containing a short introduction in to the area of error protection linear codes, basic mathematical theory of cyclic codes and also an explanation of the principle of coding and decoding. There si also a chapter describing the basic construction of encoders and decoder of cyclic codes. The practical part contains information about the programme for simulation of cyclic codes and about the technology used for its development.


Keywords: cyclic codes, coding, encoding, decoding, encoder, decoder, finite fields, Java

Děkuji vedoucímu práce panu RNDr. Ing. Miloši Krčmářovi za odborné vedení, cenné rady, návrhy a pomoc v průběhu tvorby této práce. Dále bych chtěl poděkovat panu Prof. Ing. Karlu Vlčkovi, CSc za čas, který mi věnoval a podnětné rady a informace, kterými mne obohatil. Děkuji také své rodině, za poskytnuté zázemí a podporu mého studia.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

# OBSAH

# ÚVOD

Narůstající informatizace společnosti přináší digitalizaci informací, papírové dokumenty jsou nahrazovány elektronickými, digitalizuje se telefonní komunikace, rozhlasové a televizní vysílání, analogový zápis zvuku, obrazu a videa je rychle nahrazován digitálním. Tento svět počítačů a digitálních komunikací vděčí za svou existenci a fungování výsledkům mnoha vědních oborů.

V roce 1948 vyšel článek Claude Shannona – *A Mathematical Theory of Communication*, který položil základy dvou matematických teorií - teorie informace a teorie kódování. Teorie informace je matematickou teorií, která se zabývá zákonitostmi přenosu a zpracování informace. Teorie kódování se zabývá tím, jak rychle a spolehlivě přenášet informace z jednoho místa na druhé a jak najít takový způsob zápisu informace, který by umožnil dosáhnout hranic stanovených teorií informace. Samotná teorie kódování je pěkným příkladem využití algebraických metod na řešení praktických problémů vznikajících při zpracování informace. Při přenosu dat pomocí komunikačního kanálu a při jejich uchovávání na paměťových médiích vznikají chyby, které mohou zůstat neodhaleny a při automatickém zpracování mohou způsobovat vážné problémy. Teorie kódování poskytuje řešení v podobě bezpečnostních kódů, které jsou schopné odhalovat a opravovat chyby. Výběr vhodného bezpečnostního kódu bývá kompromisem mezi stupněm požadovaného zabezpečení, jednoduchostí a cenou realizace příslušných kódových systémů, přičemž je snaha dosáhnout takového kódu, aby poměr počtu kontrolních symbolů k celkovému počtu symbolů v dané zprávě (kódovém slově) byl co nejmenší.

Úsilí o nalezení kódových systémů pro kódy s dlouhými kódovými slovy, které by byly implementovány co nejmenším počtem obvodových prvků, vedlo ke studiu tzv. cyklických kódů. [1] Základní informace o těchto zajímavých kódech, matematické teorii, kterou jsou podloženy, jejich vlastnostech a způsobu použití budou předmětem následujících kapitol.

# I. TEORETICKÁ ČÁST

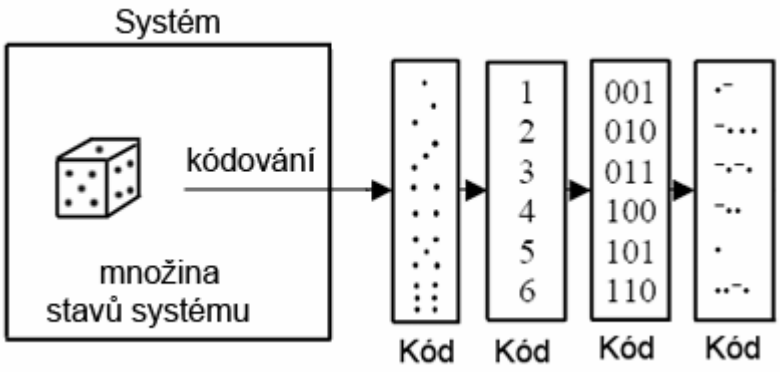
# 1 KÓDY A KÓDOVÁNÍ

## 1.1 Úvod

Činnost digitálních zařízení je z hlediska technické realizace založena na práci s binární soustavou. Pro člověka je poměrně obtížné v této soustavě provádět běžné početní operace, proto mezi symboly přirozeného jazyka (zvuky, písmena) a informacemi, které využívají digitální zařízení, existují jistá převodní pravidla – kódy.

Kód představuje množinu symbolů, kterými lze popsat různé stavy daného systému. Může být vyjádřen např. prostřednictvím tabulky, či algoritmem, který představuje sadu dohodnutých pravidel.

Kódování je pak činnost, při které převádíme jeden kód na druhý. Tento převod je realizován pomocí tabulky nebo vhodným algoritmem. Vztah mezi systémem, kódováním a kódem je znázorněn na obrázku (Obr. 1).




*Obr. 1. Vztah mezi systémem, kódem a kódováním*

Pro názornou ilustraci je jako systém zvolena běžná hrací kostka. Po hodu kostkou se může tento systém nacházet v jednom ze šesti možných stavů. Každý stav (poloha) kostky je vyjádřen tečkovým kódem. Pro zaznamenání informace o počtu teček však zřejmě zvolíme arabské číslice 1 až 6, čímž provedeme kódování do jiné množiny symbolů, která vyjadřuje rovněž aktuální stav kostky. Takovýmto způsobem bychom mohli zakódovat konkrétní stav systému pomocí různých kódů a provádět dále jejich vzájemný převod prostřednictvím kódování.

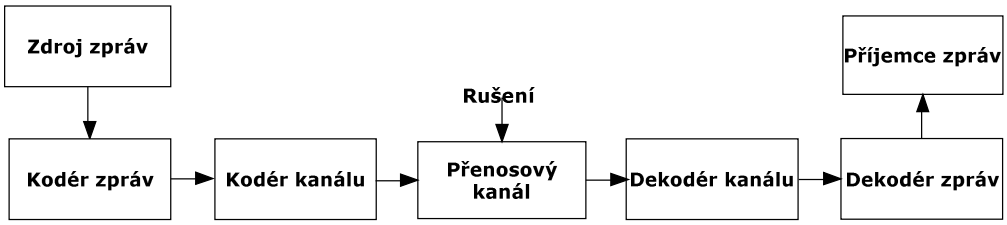
Příklad pro stav systému: ⁘ → 5 → 101 → •

Každý zdroj informace produkuje zprávu v nějaké abecedě. V teorii informace abecedou nazýváme libovolnou konečnou množinu dobře vzájemně rozlišitelných objektů, které označujeme jako znaky (symboly, popř. písmena) dané abecedy. [2]

Slovem, které muže vzniknout nad danou abecedou, je libovolná konečná uspořádaná posloupnost znaků této abecedy, nebo také každý samostatný znak této abecedy.

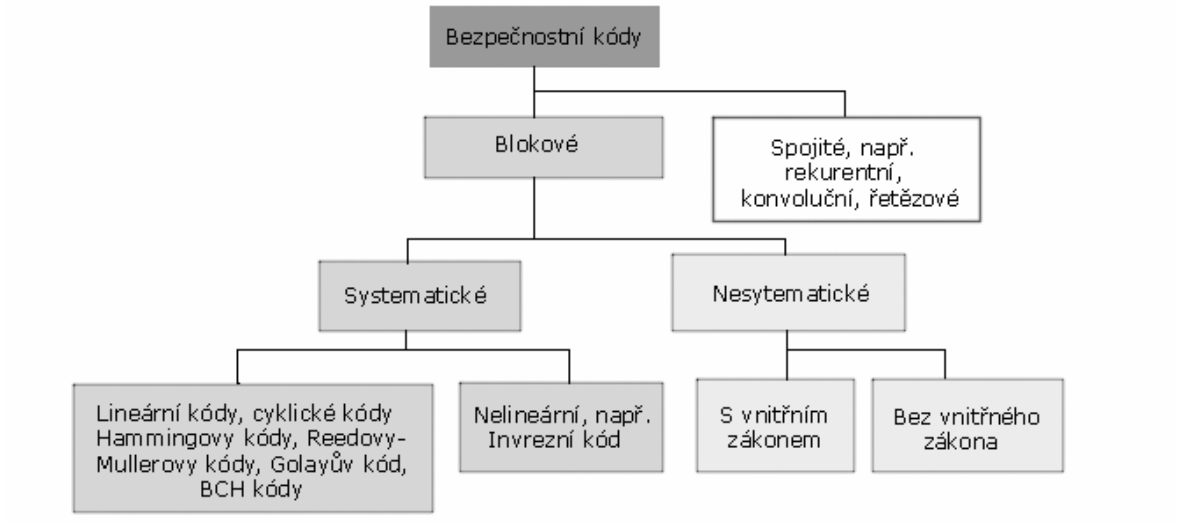
## 1.2 Bezpečnostní kódy

Při přenosu zakódovaných zpráv často dochází k chybám, které vznikají rušením v přenosovém kanálu. Kódování je založeno na principu vložení kodéru před přenosový kanál a vložení dekodéru za něj. Vhodnou volbou kódu pak dochází k ošetření chyb.




*Obr. 2. Schéma komunikační soustavy*

Bezpečnostní kód může být detekční nebo samoopravný. Detekční kód slouží pro detekci chyb, samoopravný kód chyby navíc opravuje. Některé typy bezpečnostních kódů a jejich rozdělení je uvedeno na obrázku (Obr. 3).



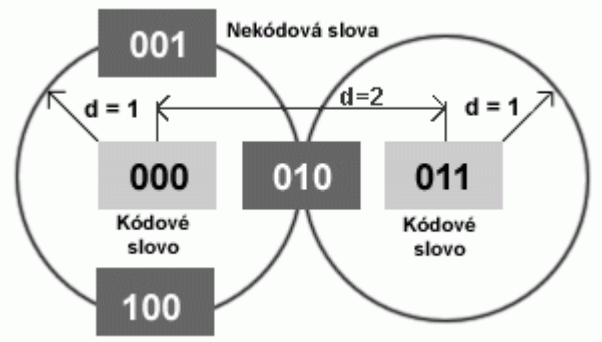

*Obr. 3. Rozdělení bezpečnostních kódů*

### 1.2.1 Hammingova vzdálenost

Hlavní myšlenkou kódového zabezpečení je to, že kódová slova musí být podle nějakého kritéria rozpoznatelná. Z tohoto důvodu je zaveden pojem vzdálenosti mezi kódovými slovy. Dvě kódová slova, pokud mají být porovnatelná musí mít stejnou délku. [1]

Uvažujeme-li dvě kódová slova $c_1$ a $c_2$, která mají stejnou délku, je jejich *Hammingova vzdálenost* definována jako počet znaků, ve kterých se tato dvě kódová slova liší. Symbolicky se značí jako $d(c_1, c_2)$ = *počet odlišností*.

*Minimální Hammingova vzdálenost d* je definována jako nejmenší vzdálenost dvou kódových slov daného kódu. Vztah mezi *minimální Hammingovou vzdáleností*, kódovými a nekódovými slovy je znázorněn na obrázku (Obr. 4).




*Obr. 4. Binární slova kódu délky 3 bity*

### 1.2.2 Detekční a korekční schopnosti kódu

Pod pojmem detekční schopnosti kódu rozumíme schopnost kódu objevovat - detekovat chyby vzniklé při jeho přenosu. Kód objevuje (detekuje) $\alpha$-násobné chyby tehdy, platí li:

$$d \geq \alpha + 1 \qquad (1)$$

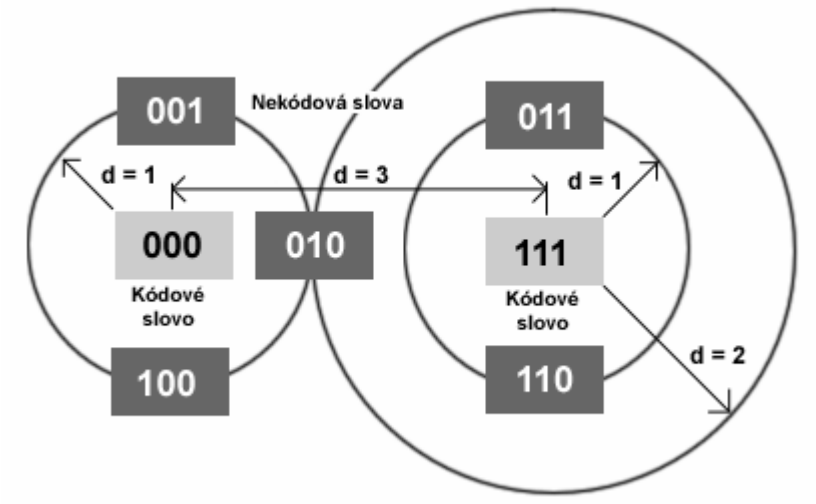
Pod pojmem korekční schopnosti kódu rozumíme schopnost opravit - korigovat chyby vzniklé při jeho přenosu. Kód opravuje (koriguje) $\beta$-násobné chyby tehdy, platí li:

$$d \geq 2\beta + 1 \qquad (2)$$

***Příklad:***

Uvažujeme-li kód s $d = 1$ a kódové slovo $c_1 = 000$, které se vlivem rušení změnilo na $c_1` = 010$, je nemožné vyhodnotit, zda je přijatá zpráva chybná či správná, neboť $c_1` = 010$ je opět platným kódovým slovem daného kódu. V případě, že kód má $d = 2$, bylo by přijaté

kódové slovo $c_1^{`} = 010$ vyhodnoceno jako chybné, neboť mu není přiřazen význam platného kódového slova. Jelikož však toto neplatné kódové slovo mohlo vzniknout z více slov platných, není možné určit přesnou polohu chyby nutnou pro její opravu. Tato situace je zobrazena na obrázku (Obr. 4). Chceme-li chybu nejen detekovat, ale i opravit, je v tomto případě nutné uvažovat kód s $d = 3$. Přijaté kódové slovo $c_1^{`} = 010$ je pak vzdáleno o $d = 1$ od původního kódového slova a o $d = 2$ od dalšího platného kódového slova $c_2 = 111$, chybu lze tedy nejen detekovat, ale i opravit. Obrázek (Obr. 5) znázorňuje situaci při $d = 3$.




*Obr. 5. Kódová slova s minimální Hammingovou vzdáleností d=3.*

## 1.3 Lineární blokové kódy

Jelikož cyklické kódy patří ke kódům lineárním, je v této kapitole nastíněna základní problematika lineárních kódů.

*Lineární blokové kódy* patří do skupiny systematických kódů. Systematický kód má slovo skládající se z oddělené informační části a kontrolní části. Lineární blokové kódy se vyznačují tím, že libovolná lineární kombinace kódových slov je opět kódovým slovem. Obyčejně se značí také jako *(n, k)* kódy, kde *n* je délka kódu a *k* počet informačních znaků v kódové kombinaci.

Při použití systematického binárního kódu lze rozlišit část, která je původním informačním slovem (informační bity) a část, která byla přidána z důvodu kontroly nebo zabezpečení

informace (zabezpečovací nebo kontrolní bity). Je zřejmé, že počet zabezpečujících bitů $r$ odpovídá $r = n - k$.

Označení, blokový kód, znamená, že délka všech kódových slov je stejná a odpovídá délce kódu $n$.

Pro sčítání a násobení prvků (bitů) dvojstavového signálu je nutné zavést speciální operace. Název pro tyto operace je *sčítání* a *násobení modulo 2*, používá se zkratka mod*2*. Pro sčítání mod*2*, které se označuje také symbolem $\oplus$ platí tabulka (Tab. 1), pro násobení platí tabulka (Tab. 2).

*Tab. 1. Sčítání modulo 2*

| | | | | |
|---|---|---|---|---|
| *A* | 1 | 0 | 1 | 0 |
| *B* | 1 | 1 | 0 | 0 |
| $\boldsymbol{A \oplus B}$ | **0** | **1** | **1** | **0** |

*Tab. 2. Násobení modulo 2*

| | | | | |
|---|---|---|---|---|
| *A* | 1 | 0 | 1 | 0 |
| *B* | 1 | 1 | 0 | 0 |
| $\boldsymbol{A \times B}$ | **1** | **0** | **0** | **0** |

### 1.3.1 Generující matice

Pro nalezení kódových slov lineárního binárního kódu je výhodné najít tzv. *bázi kódu*. *Báze kódu* je minimální množina vektorů, z nichž lze všechny ostatní vektory (slova) daného lineárního kódu odvodit povolenými operacemi. Z kódových slov báze je pak možné sestavit *generující matici kódu G*. Pro generující matici o rozměru ($n$ x $k$) musí platit, že každý její řádek je platným kódovým slovem, jednotlivé řádky jsou lineárně nezávislé a každé kódové slovo je lineární kombinací řádků.

Vybraná nezabezpečená kódová slova mohou tvořit jednotkovou matici *E*. V případě, že tato kódová slova doplníme o zabezpečující prvky vytvořené algoritmem daného kódování, je možné sestavit tzv. *generující matici systematického kódu*:

$$G = [E \mid R] \tag{3}$$

kde *E* je jednotková matice řádu *k* a *R* je matice zabezpečujících prvků. Kódování, jehož výsledkem je zabezpečené kódové slovo *v*, provedeme součinem nezabezpečeného slova *u* s generující maticí:

$$v = u \cdot G \tag{4}$$

### 1.3.2 Kontrolní matice

Po přenosu zakódovaného slova se chceme přesvědčit o správnosti přenosu. K tomu potřebujeme tzv. *kontrolní matici H*, která má *r* řádků a *n* sloupců a odvozuje se z generující matice *G* podle vztahu:

$$H = [-R^T \mid E] \tag{5}$$

Kontrolu správnosti přeneseného slova uskutečníme tak, že vektor přenesené kódového slova vynásobíme transponovanou kontrolní maticí $H^T$, která vznikne z *H* otočením kolem hlavní diagonály, a má tedy rozměr (*n* x *r*). Výsledkem násobení je tzv. *chybový vektor* neboli *syndrom S*. Je-li tento vektor nulový, pak byl přenos bezchybný. Obsahuje-li alespoň jednu jedničku, pak během přenosu nastala chyba.

***Příklad**:*

Přijatá kódová slova: *01111, 01101*

Kontrolní matice: $H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$

$$S = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad S = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

*Bezchybný přenos* *Přenos s chybou*

Podrobnější vysvětlení problematiky lineárních blokových kódů je uvedeno v literatuře [1], [2], [3], [4] a [5].

# 2 CYKLICKÉ KÓDY

## 2.1 Úvod

Úsilí o nalezení kódových systémů pro kódy s dlouhými kódovými slovy, které by byly implementovány co nejmenším počtem obvodových prvků, vedlo ke studiu tzv. cyklických kódů. [1]

Cyklické kódy jsou rozsáhlou třídou lineárních kódů a patří mezi nejmodernější bezpečnostní kódy, které se osvědčily v praxi. Jejich specifická vlastnost, podle které se též nazývají, spočívá v tom, že cyklickou záměnou prvků použitého kódového slova vzniká opět platné kódové slovo. Podle způsobu manipulace s kódovým slovem se označují také jako kódy polynomické. Hlavní výhodou je snadná obvodová realizace, kdy se posloupnost symbolů zpracovává pomocí zpětnovazebních posuvných registrů, proto jsou tyto kódy oblíbené především v situacích, kdy je třeba pro účely kódování zkonstruovat jednoúčelové technické zařízení. Široké uplatnění nachází pro své velmi dobré detekční schopnosti jak osamělých chyb, tak i shluků chyb. Následující kapitoly jsou zaměřeny na cyklické kódy pro opravu jedné chyby.

Z matematického hlediska je teorie cyklických kódů založena na popisu vlastností *konečných těles*.

## 2.2 Matematický základ

### 2.2.1 Grupa, okruh, těleso

Lineární kódy se označují také jako kódy grupové. Toto označení je odvozeno od pojmu *grupa*. Množinu *G*, na níž je definována binární operace # nazýváme *grupou*, platí li vztahy:

$$x\#(y\#z)=(x\#y)\#z \quad \textit{pro všechny } x, y, z, \in G \text{ (asociativnost)} \tag{6}$$

$$1\#x=x\#1=x \quad \textit{pro všechny } x \in G \text{ (neutrální prvek)} \tag{7}$$

$$x\#x^{-1}=x^{-1}\#x=1 \quad \textit{pro všechny } x \in G \text{ (inverzní prvky)} \tag{8}$$

Platí-li pro *G* také komutativní zákon:

$$y\#x=x\#\mathrm{y} \tag{9}$$

říkáme, že grupa *G* je *komutativní grupa.*

Operace # může být provedena na jakýchkoliv dvou prvcích grupy, přičemž obdržíme třetí prvek, který je také součástí grupy.

Algebraický okruh *R* je množina, v níž jsou definovány dvě binární operace + a · (sčítání a násobení). *R* je komutativní grupa vzhledem k operaci + a vzhledem k operaci · pro ni platí asociativní zákon (6). Pro každé tři prvky $x, y, z \in R$ platí distributivní zákon:

$$x(y+z)=x\,y+x\,z \quad \text{a} \quad (x+y)z=x\,z+y\,z \tag{10}$$

Okruh lze označit jako komutativní, v případě, že pro něj platí komutativní zákon (9). Speciálním typem okruhu je *okruh mnohočlenů R[x]*.

*Těleso T*, je komutativní okruh, pro nějž platí existence jednotkového prvku vzhledem k násobení a existuje prvek inverzní ke každému nenulovému prvku. Existence inverzního prvku umožňuje definovat operaci dělení jako násobení inverzním prvkem.

### 2.2.2 Konečná tělesa

Je-li těleso *T* tvořeno konečnou množinou prvků, pak *T* je *konečné těleso*. Příkladem konečného tělesa, které má velký význam v oblasti kódování je těleso označené jako $Z_{p.}$

Konečné těleso $Z_p$ je algebraická struktura, která je tvořená zbytky po dělení celých kladných čísel prvočíslem *p*. Pro vyjádření problémů kódování je velmi důležité konečné těleso s $p = 2$, tj. $Z_2$ , tvořené množinou *{0, 1}*. Nad $Z_p$ je možno definovat *okruh mnohočlenů* $Z_p\,[x]$, tzn. že i pro $Z_2$ okruh $Z_2\,[x]$.

Tělesa označená jako *Galoisova tělesa* $GF(p^r)$, vznikají rozšířením konečného tělesa $Z_p$ stupněm *r*. Galoisovo těleso tedy obsahuje konečný počet $p^r$ prvků. Pravidla pro aritmetické operace mezi prvky tělesa $GF(p^r)$ jsou určena operacemi *modulo p*. Operace *modulo 2* již byla uvedena tabulkách (Tab. 1) a (Tab. 2), a splňuje všechny pravidla pro těleso $GF(2)$.

***Příklad:***

Aritmetické operace pro $GF(3)$:

*Tab. 3. Sčítání v GF(3)*

| + | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 2 | 0 |
| 2 | 2 | 0 | 1 |

*Tab. 4. Násobení v GF(3)*

| × | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 1 |

Z tabulky (Tab. 4) lze určit, že inverzní prvek k prvku 1 je 1 a inverzní prvek k prvku 2 je 2.

Jiný, obecnější přístup pro nalezení inverzních prvků, který je pak možné použít na tělesa s větším počtem prvků, je nalezení tzv. *primitivního prvku* tělesa. Prvek je označen jako primitivní, v případě, že všechny nenulové prvky tělesa mohou být vyjádřeny pomocí mocnin tohoto prvku.

***Příklad:***

V $GF(7)$ je primitivním prvkem číslo 3, neboť pro něj platí:

$$3^0 = 1 \qquad 3^1 = 3 \qquad 3^2 = 2$$
$$3^3 = 6 \qquad 3^4 = 4 \qquad 3^5 = 5$$

Hodnoty pro vyšší mocniny čísla 3 se dále cyklicky opakují: $3^6 = 1$. Násobení nyní můžeme provádět prostřednictvím mocnin čísla 3, např. 6 x 2 = $3^3 \text{x} 3^2$ = $3^5$ = 5. Z tohoto důvodu tedy můžeme najít i inverzní prvek k prvku $3^i$, tak, že inverzní prvek $3^{-i} = 3^{6-i}$. Pro prvky z předchozího příkladu platí, že inverzní prvek k prvku 4 ($3^4$) je 2 ($3^2$), inverzní prvek k prvku 5 ($3^5$) je 3 ($3^1$) a inverzní prvek k prvku 6 ($3^3$) je 6 ($3^3$).

### 2.2.3 Konečná tělesa tvořená mnohočleny

Podobně, jako byly v předchozí části všechny nenulové prvky tělesa vyjádřeny prostřednictvím *primitivního prvku*, lze u tělesa tvořeného mnohočleny najít jeho prvky prostřednictvím tzv. *ireducibilního mnohočlenu*.

Říkáme, že mnohočlen *f(x)* je *ireducibilní* nad tělesem *T*, jestliže neexistuje rozklad *f(x)=a(x).b(x)*, kde mnohočleny *a(x)* a *b(x)* nad *T* jsou nenulové a mají nižší stupeň než je stupeň mnohočlenu *f(x)*.

Nechť *f(x)*, je ireducibilní mnohočlen stupně *m* nad *GF(p)*. Pak množina mnohočlenů stupně menšího než *m*, s operacemi sčítání, odečítání, násobení a dělení s aritmetikou mod*f(x)* tvoří konečné těleso *T*, které má $p^m$ prvků. [1]

Za předpokladu, že $\alpha$ je kořen ireducibilního mnohočlenu *f(x)*, můžou být prvky tělesa chápány jako mnohočleny mod*f(x)*, nebo jako mnohočleny v mocninách $\alpha$.

Uvažujeme-li např. ireducibilní mnohočlen $f(x)= x^3 + x + 1$ s kořenem $\alpha$, platí vztah, $\alpha^3 + \alpha + 1 = 0$. Následující příklad uvádí konstrukci konečného binárního tělesa s osmi prvky.

***Příklad*:**

Konečné binární těleso $GF\left(2^3\right) \equiv GF(8)$ je tvořeno osmi binárními čísly o délce 3 bity. Tyto binární čísla je možné vyjádřit pomocí mnohočlenů, tak jak to popisuje následující tabulka (Tab. 5).

*Tab. 5. Prvky GF(8) vyjádřené mnohočleny*

| | | | |
|---|---|---|---|
| 000 | 0 | 011 | $x+1$ |
| 001 | 1 | 110 | $x^2 + x$ |
| 010 | $x$ | 111 | $x^2 + x + 1$ |
| 100 | $x^2$ | 101 | $x^2 + 1$ |

Pro tyto prvky je definována operace sčítání a násobení a vzhledem k vlastnostem multiplikativní grupy je zajištěna cykličnost. Vynásobíme-li tedy jakékoliv dva prvky, výsledkem musí být prvek již napsaný.

Uvažujeme-li např. slovo *010* a *011* je jejich součin vyjádřený mnohočleny $(x+1).x = x^2 + x$, což odpovídá slovu *110*. Pokud uvažujeme slova *110* a *010* je součin $(x^2 + x).x = x^3 + x^2$. Takové slovo však neodpovídá žádnému ze slov v (Tab. 5), neboť nevíme co přestavuje prvek $x^3$. Prvky konečného tělesa jsou chápány jako mnohočleny mod*f(x)*. V tomto případě lze tedy při uvažovaném ireducibilní mnohočlenu $f(x) = x^3 + x + 1$, určit zbytek po dělení $x^3 : x^3 + x + 1$, který odpovídá mnohočlenu $x+1$. Prvek $x^3$ je tedy vyjádřen jako $x^3 = x + 1$ a slovo $x^3 + x^2$ lze určit jako $x^3 + x^2 = x^2 + x + 1$. Podobným způsobem je možné určit násobky ostatních prvků *GF(8)*. Následující tabulka (Tab. 6) obsahuje všechny možné násobky slov v *GF(*8*)*. V tabulce si

je též možné povšimnout výsledků s hodnotou 1, které tak představují součin navzájem inverzních prvků.

*Tab. 6. Násobení prvků v GF(8) s aritmetikou modulo* $x^3 + x + 1$

| × | 0 | 1 | $x$ | $x+1$ | $x^2$ | $x^2+1$ | $x^2+x$ | $x^2+x+1$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | $x$ | $x+1$ | $x^2$ | $x^2+1$ | $x^2+x$ | $x^2+x+1$ |
| $x$ | 0 | $x$ | $x^2$ | $x^2+x$ | $x+1$ | 1 | $x^2+x+1$ | $x^2+1$ |
| $x+1$ | 0 | $x+1$ | $x^2+x$ | $x^2+1$ | $x^2+x+1$ | $x^2$ | 1 | $x$ |
| $x^2$ | 0 | $x^2$ | $x+1$ | $x^2+x+1$ | $x^2+x$ | $x$ | $x^2+1$ | 1 |
| $x^2+x$ | 0 | $x^2+x$ | 1 | $x^2$ | $x$ | $x^2+x+1$ | $x+1$ | $x^2+x$ |
| $x^2+x$ | 0 | $x^2+x$ | $x^2+x+1$ | 1 | $x^2+1$ | $x+1$ | $x$ | $x^2$ |
| $x^2+x+1$ | 0 | $x^2+x+1$ | $x^2+1$ | $x$ | 1 | $x^2+x$ | $x^2$ | $x+1$ |

Jak již bylo uvedeno, konečná tělesa lze konstruovat pomocí ireducibilního mnohočlenu nebo prostřednictvím mocnin prvku $\alpha$, který představuje kořen ireducibilního mnohočlenu. Pro ireducibilní mnohočlen $f(x) = x^3 + x + 1$ s kořenem $\alpha$ tedy platí $\alpha^3 + \alpha + 1 = 0$ a $\alpha^3 = \alpha + 1$. Dále je pak možné vyjádřit $\alpha^4 = \alpha^3\alpha = (\alpha + 1)\alpha = \alpha^2 + \alpha$ a obdobným způsobem člen $\alpha^5$ a $\alpha^6$.

V následující tabulce (Tab. 7) jsou vyjádřeny pomocí mocnin $\alpha$ mnohočleny *GF(8)*.

*Tab. 7. Tabulka mnohočlenů GF(8)*

| 1 | 2 | 3 | | | | |
|---|---|---|---|---|---|---|
| 000 | 0 | | | | | 0 |
| 001 | 1 | | | | | 1 |
| 010 | $\alpha$ | | | $\alpha$ | | |
| 100 | $\alpha^2$ | $\alpha^2$ | | | | |
| 011 | $\alpha^3$ | | | $\alpha$ | + | 1 |
| 110 | $\alpha^4$ | $\alpha^2$ | + | $\alpha$ | | |
| 111 | $\alpha^5$ | $\alpha^2$ | + | $\alpha$ | + | 1 |
| 101 | $\alpha^6$ | $\alpha^2$ | | | + | 1 |

V prvním sloupci jsou uvedeny binární hodnoty, ve druhém jsou seřazeny hodnoty $\alpha$ podle velikosti mocnin, přičemž hodnota $\alpha^7 = 1$ a ve třetím sloupci jsou odpovídající mnohočleny. Pomocí druhého sloupce lze snadno provádět operace násobení a dělení, pomocí třetího operaci sčítání. Hledání inverzního prvku k $\alpha^i$ je možné provést jako $\alpha^{-i} = \alpha^{7-i}$.

Podrobnější vysvětlení problematiky konečných těles je uvedeno v literatuře [1], [3] a [5].

## 2.3 Konstrukce cyklického kódu

Lineární kód se nazývá *cyklický*, obsahuje-li s každým kódovým slovem $a = a_0 a_1 a_2 ... a_{n-1}$ také kódové slovo $a` = a_{n-1} a_0 a_1 ... a_{n-2}$ , jež vzniklo cyklickým posuvem ze slova $a$ .

Pro snadný matematický popis cyklického kódu se používá zápis kódových slov prostřednictvím mnohočlenů. Přiřazení kódových slov a mnohočlenů probíhá podle následujícího předpisu:

$$a_0 a_1 a_2 ... a_{n-1} \leftrightarrow a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1} \qquad (11)$$

Uvažujeme-li kódové slovo $a(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$, pak cyklický posun prvků kódového slova odpovídá násobení mnohočlenu proměnnou *x*. Koeficient, který by vznikl násobením u nejvyšší mocniny, tedy $x^n$, se přesune na začátek kódového slova, neboť platí následující vztah:

$$x^n \ mod(x^n - 1) = 1 \qquad (12)$$

Dále platí $x^{n+1} = x$, $x^{n+2} = x^2$ atd. Násobením *a(x)* proměnnou *x* získáme $a(x) \cdot x = a(x) = a_{n-1} + a_0 x + a_1 x^2 + \ldots + a_{n-2} x^{n-1}$ . Je zřejmé, že při takovémto způsobu vytváření může vzniknout pouze *n* kombinací.

Pořadí, v jakém se zapisují koeficienty mnohočlenů a jednotlivých kódových slov se v různých literaturách liší. Jednou možností je zápis, kdy čteme kódové slovo „zleva doprava“. V takovém případě pak např. mnohočlen $x^3 + x^2$ odpovídá kódovému slovu *0011*. Pro příklady, které obsahuje tato práce je však zvolen druhý způsob. Mnohočlen $x^3 + x^2$ tedy odpovídá kódovému slovu *1100.*

Operace odečítání, násobení a dělení probíhají podle normálních pravidel pro práci s mnohočleny a řídí se navíc podle pravidel algebry *modulo 2*, kde operace součtu se rovná operaci rozdílu.

### 2.3.1 Generující mnohočlen

Každý cyklický *(n, k)* kód obsahuje mnohočlen *g(x)* stupně *n - k*, jenž má následující vlastnosti:

- všechna kódová slova jsou násobky mnohočlenu *g(x)*.
- mnohočleny $g(x), xg(x), ...., x^{n-k-1} g(x)$ tvoří bázi kódu.

- mnohočlen *g(x)* je dělitelem mnohočlenu $x^n - 1$ (dělí jej beze zbytku).

Tento mnohočlen se nazývá *generující mnohočlen g(x)* a musí pro něj dále platit, že $g_0 \neq 0$ a $g_{n-k} \neq 0$. Pokud by $g_0 = 0$, tak by všechna kódované slova začínala číslicí 0 a tato číslice by tedy nenesla žádnou informaci. Podmínka $g_{n-k} \neq 0$ je z podobných důvodů nutná pro to, aby poslední číslice kódovaného slova nesla nějakou informaci.[3]

Cyklickým posunem koeficientů generujícího mnohočlenu vzniká generující matice *G*:

$$G = \begin{bmatrix} g_{n-k} & g_{n-k-1} & \cdots & g_1 & g_0 & 0 & 0 & \cdots & 0 \\ 0 & g_{n-k} & g_{n-k-1} & \cdots & g_1 & g_0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & \\ 0 & 0 & \cdots & 0 & g_{n-k} & g_{n-k-1} & \cdots & g_1 & g_0 \end{bmatrix} \qquad (13)$$

Generující mnohočlen představuje tedy zjednodušený zápis generující matice. Kódová slova v generující matici tvoří bázi daného kódu.

### 2.3.2 Kontrolní mnohočlen

Podíl

$$h(x) = \frac{x^n - 1}{g(x)} \qquad (14)$$

je označen jako *kontrolní mnohočlen*. Kontrolní matici *H* cyklického kódu *(n, k)* získáme cyklickými posuvy koeficientů kontrolního mnohočlenu.

### 2.3.3 Minimální kódová vzdálenost

Má-li cyklický kód opravovat jednonásobné chyby, musí být *g(x)* alespoň trojčlenem a z množiny násobků *g(x)* je nutné vyloučit všechny dvojčleny, aby minimální kódová vzdálenost byla *d>2*. Toho se dosahuje operací $\mathrm{mod}\,(x^n - 1)$, kterou se všechny násobky mnohočlenu *g(x)* převedou na mnohočleny stupně nižšího než *n*. [1]

## 2.4 Kódování

Jedna z důležitých vlastností binárních cyklických kódů je jednoduchost kódování informačních bitů. Existují dvě kódovací metody: jedna přímá, ale nesystematická je

založena na násobení generujícím mnohočlenem a druhá, která je systematická je založena na dělení tímto mnohočlenem. [1]

### 2.4.1 Nesystematické kódování

Uvažujeme-li slovo $u = u_{k-1}...u_2u_1u_0$, nesoucí informaci, kterou chceme zakódovat, je kódování možné provést vynásobením bitů $u_{k-1}...u_2u_1u_0$ s generující maticí:

$$v = [u_{k-1}...u_2u_1u_0].\begin{bmatrix} g_{n-k} & g_{n-k-1} & \cdots & g_1 & g_0 & 0 & 0 & \cdots & 0 \\ 0 & g_{n-k} & g_{n-k-1} & \cdots & g_1 & g_0 & 0 & \cdots & 0 \\ \vdots & & & & & & & & \\ 0 & 0 & \cdots & 0 & g_{n-k} & g_{n-k-1} & \cdots & g_1 & g_0 \end{bmatrix}, \qquad (15)$$

kde $v$ je vektor výsledného kódového slova určeného k odeslání.

Vyjádříme-li slovo $u$ jako mnohočlen $u(x) = u_{k-1}x^{k-1} + ... + u_2x^2 + u_1x + u_0$, je možné kódování provést vynásobením $u(x)$ s generujícím mnohočlenem $g(x)$:

$$v(x) = (u_{k-1}x^{k-1} + ... + u_2x^2 + u_1x + u_0).(g_{n-k}x^{n-k} + ... + g_2x^2 + g_1x + g_0) \qquad (16)$$

Mnohočlen $v(x)$ stupně $n$ - 1 pak vyjadřuje opět výsledné kódové slovo.

***Příklad:***

Je dán kód (7, 4), kde informační slovo *1110*, určené k zakódování, odpovídá mnohočlenu $u(x) = x^3 + x$. Generující mnohočlen kódu je zadán jako $g(x) = x^3 + x^2 + 1$, což odpovídá binárnímu vyjádření *1101*. Kódování pak proběhne jako:

$$v = [1 \quad 0 \quad 1 \quad 0].\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} = [1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0]$$

nebo

$$v(x) = (x^3 + x).(x^3 + x^2 + 1) = x^6 + x^5 + x^3 + x^4 + x^3 + x = x^6 + x^5 + x^4 + x$$

Výsledný mnohočlen kódového slova $v(x) = x^6 + x^5 + x^4 + x$ odpovídá kódovému slovu *v =1110010*. Jelikož není z tvaru tohoto slova viditelné, jaké informační bity byly zakódovány (*1010*), a jaké bity jsou zabezpečující, je zřejmé, že takové kódování je nesystematické.

### 2.4.2 Systematické kódování

Při systematickém kódování je využíváno operace dělení mnohočlenů. Prvním krokem je doplnění posloupnosti informačních bitů o $n-k$ nulových míst. Toho dosáhneme vynásobením informačního mnohočlenu *u(x)* členem $x^{n-k}$, čímž získáme prostor pro připojení $n-k$, zabezpečujících prvků. Takto upravený informační mnohočlen *u(x)* poté vydělíme generujícím mnohočlenem *g(x)*.

$$\frac{x^{n-k}.u(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \qquad (17)$$

Při dělení je vypočten podíl *q(x)*, který nebude nijak využit a zbytek po dělení vyjádřený mnohočlenem *r(x)*. Mnohočlen *r(x)* může být nejvýše stupně $n-k-1$, neboť dělitel je stupně $n-k$. Zabezpečené kódové slovo pak představuje mnohočlen $v(x)= x^{n-k}.u(x) + r(x)$. Jelikož mnohočlen *v(x)* vznikl součtem informačního mnohočlenu se zbytkem po dělení informačního mnohočlenu generujícím mnohočlenem, je zřejmé, že *v(x)* je beze zbytku dělitelný generujícím mnohočlenem *g(x)*.

Skupina zabezpečujících prvků, vyjádřených mnohočlenem *r(x)*, bývá také označována zkratkou *CRC – Cyclic Redundancy Check*.

Vysílaná zabezpečená zpráva má tedy tvar:

| Informační část | CRC |
|---|---|

***Příklad:***

Je dán kód (7, 4), kde informační slovo *1110*, určené k zakódování, odpovídá mnohočlenu $u(x)= x^3 + x^2 + x$. Generující mnohočlen kódu je zadán jako $g(x)= x^3 + x^2 + 1$, což odpovídá binárnímu vyjádření *1101*. Úpravou $x^{n-k} \cdot u(x) = x^3 \left(x^3 + x^2 + x\right)$ získáme mnohočlen $x^6 + x^5 + x^4$, se kterým budeme dále pracovat. Výpočet kontrolních bitů probíhá následujícím způsobem:

$$\begin{array}{l} x^6 + x^5 + x^4 \quad : \quad x^3 + x^2 + 1 = x^3 + x \\ -(x^6 + x^5 + x^3) \\ \qquad x^4 + x^3 \\ \qquad -(x^4 + x^3 + x) \\ \qquad\qquad x = r(x) \end{array}$$

nebo

$$\begin{array}{r} 1110000 \\ \oplus\ 1101 \\ \hline 0011000 \\ \oplus\ \ 1101 \\ \hline 010 \end{array}$$

Zbytek po dělení je zapsán jako mnohočlen $x$, který odpovídá slovu *010*. Zabezpečené kódové slovo bude tedy odpovídat součtu $v(x) = x^{n-k} \cdot u(x) + r(x) = x^6 + x^5 + x^4 + x$, což odpovídá slovu *1110010*. Jelikož je mnohočlen zbytku po dělení vždy nižšího stupně než generující mnohočlen, znamená to, že *u(x)* a *r(x)* se při sčítání neprostupují a ve výsledném mnohočlenu *v(x)* lze rozlišit informační a zabezpečující bity.

Vysílaná zpráva má tedy požadovaný tvar:

| Informační část | CRC | = | 1110 | 010 |
|---|---|---|---|---|

## 2.5 Dekódování

Na přijímací straně obdržíme mnohočlen přenesené zprávy *w(x)* – ten může být rozdílný od mnohočlenu odeslaného kódového slova *v(x)*, je proto nutné otestovat jej na přítomnost chyb. Tento test spočívá ve vydělení přijatého slova *w(x)* daným generujícím mnohočlenem *g(x)*. Je-li *w(x)* dělitelné *g(x)* beze zbytku, pak byl přenos bezchybný a platí rovnost *w(x)=v(x)*. V případě, že vznikne nenulový zbytek označený jako mnohočlen *s(x)*, pak nastala při přenosu chyba, která byla tímto způsobem detekována. Mnohočlen *s(x)* se nazývá *syndrom přijatého slova w(x)*.

Při opravě chyb je nutné vycházet z tzv. *chybového mnohočlenu e(x)*. Binární vyjádření chybového mnohočlenu bude obsahovat hodnotu 1 právě v místě výskytu chyby, tedy v místě odlišností *w(x)* a *v(x)*, kdežto na ostatních místech bude hodnota 0. Pro mnohočlen přijaté zprávy platí:

$$w(x) = v(x) + e(x) \tag{18}$$

V případě bezchybného přenosu, kdy je mnohočlen *e(x)* nulový, je nulový také syndrom *s(x)*. Vznik nenulového syndromu *s(x)* je tedy závislý na nenulovosti chybového mnohočlenu *e(x)* a platí, že přijeté slovo *w(x)* má stejný syndrom *s(x)* (zbytek pod dělení mnohočlenem *g(x)* ), jako je syndrom chybového mnohočlenu *e(x)*.

V dalším kroku stačí tedy provést dělení $\frac{e(x)}{g(x)} = s(x)$ pro všechny možné mnohočleny *e(x)*, tj. pro všechny možné kombinace uvažovaných chyb. Kombinace *e(x)*, která bude mít stejný syndrom jako je syndrom přijatého slova *w(x)*, bude určovat příslušný chybový mnohočlen a tedy i pozici chyb. Oprava se pak provede sečtením mnohočlenu *e(x)* s přijatým mnohočlenem *w(x)*.

***Příklad:***

Je dán kód (7, 4), s generujícím mnohočlenem $g(x)= x^3 + x^2 + 1$. Za použití systematického kódování bylo vytvořeno a odesláno slovo *1110010*, odpovídající mnohočlenu $v(x)= x^6 + x^5 + x^4 + x$. Na přijímací straně bylo přijato slovo *1010010*, které odpovídá mnohočlenu $w(x)= x^6 + x^4 + x$. Dekódování probíhá následujícím způsobem:

$$
\begin{array}{l}
x^6 + x^4 + x \quad : \quad x^3 + x^2 + 1 = x^3 + x^2 + 1 \\
-(x^6 + x^5 + x^3) \\
\qquad x^5 + x^4 + x^3 + x \\
\qquad -(x^5 + x^4 + x^2) \\
\qquad\quad x^3 + x^2 + x \\
\qquad\quad -(x^3 + x^2 + 1) \\
\qquad\qquad x + 1 = s(x)
\end{array}
$$

nebo

$$
\begin{array}{r}
1\,0\,1\,0\,0\,1\,0 \\
\oplus\ 1\,1\,0\,1 \\
\hline
0\,1\,1\,1\,0\,1\,0 \\
\oplus\ \ 1\,1\,0\,1 \\
\hline
0\,0\,0\,1\,1\,1\,0 \\
\oplus\ \ \ \ 1\,1\,0\,1 \\
\hline
0\,1\,1
\end{array}
$$

Výpočet syndromu chybového mnohočlenu *e(x)*:

$$
\begin{array}{l}
x^5 \quad : \quad x^3 + x^2 + 1 = x^2 + x \\
-(x^5 + x^4 + x^2) \\
\qquad x^4 + x^2 \\
\qquad -(x^4 + x^3 + x) \\
\qquad\quad x^3 + x^2 + x \\
\qquad\quad -(x^3 + x^2 + 1) \\
\qquad\qquad x + 1 = s(x) \quad \rightarrow
\end{array}
$$

*Tab. 8. Tabulka syndromů*

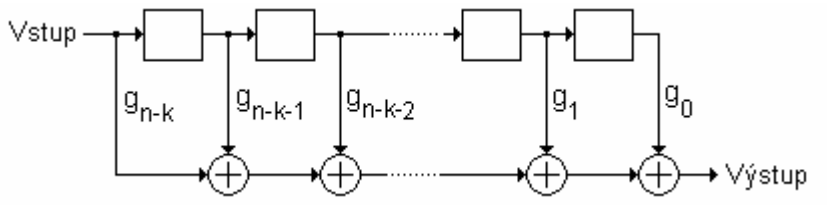
| pozice chyby | Syndrom |
|---|---|
| bez chyby | 000 |
| 0 | 001 |
| 1 | 010 |
| 2 | 100 |
| 3 | 101 |
| 4 | 111 |
| 5 | 011 |
| 6 | 110 |

Byl zjištěn syndrom přijatého slova *s(x)*=*x*+1, který odpovídá syndromu chybového mnohočlenu $x^5$. Chyba při přenosu tedy vznikla na páté pozici (čísluje se od nuly). Oprava chyby proběhne sečtením $(x^6 + x^4 + x) + x^5 = x^6 + x^5 + x^4 + x = v(x)$.

## 2.6 Realizace kodéru a dekodéru

Princip zabezpečení pospaný v předchozích kapitolách lze jednoduše realizovat pomocí kruhového posuvného registru se zpětnými vazbami a vhodně zapojenými sčítačkami *modulo 2*. Podle typu zapojení těchto obvodů lze elektronicky provádět operaci násobení nebo dělení, což umožňuje sestavit kodér a dekodér jak nesystematického, tak i systematického kódu.

V následujícím popisu bude věnována pozornost především obvodům, které jsou využívány pro tvorbu systematického kódu a umožňují tedy operaci dělení mnohočlenů. Obvod, který lze využít pro vytváření kódu nesystematického, je znázorněn na obrázku (Obr. 6).




*Obr. 6. Obvod pro násobení mnohočlenem g(x)*

### 2.6.1 Kodér systematického kódu

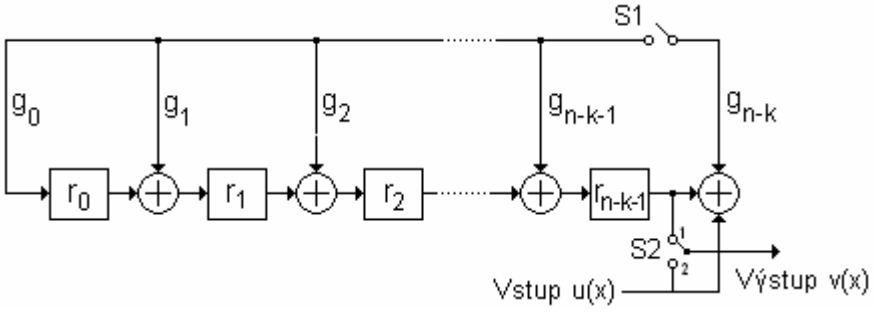
Pro vytvoření systematického kódu je nutné provést dělení informačního mnohočlenu generujícím mnohočlenem *g(x)*. Posuvný registr, kde se tato operace uskutečňuje je označován jako *dělička* mod*g(x)*. Pro odvození zapojení děličky postupujeme podle následujícího návodu, popsaného v lit. [7]:

Posuvný registr má *n - k* paměťových buněk, což odpovídá řádu generujícího mnohočlenu *g(x)*.

Místa zařazení zpětných vazeb a umístění sčítaček mod*2* jsou následující:

Z generujícího mnohočlenu *g(x)* zjistíme, před kterými členy je sčítací znaménko, a před ty samé členy v posuvném registru zařadíme sčítačky mod*2* se zpětnými vazbami. Znaménko $\oplus$ je také před členem nejvyššího řádu generujícího mnohočlenu, avšak ten již nemá svou buňku v posuvném registru.

Popsaný princip je znázorněn na obrázku (Obr. 7).




*Obr. 7. Kodér systematického cyklického kódu*

Kodér pracuje následujícím způsobem, popsaným v lit. [7]:

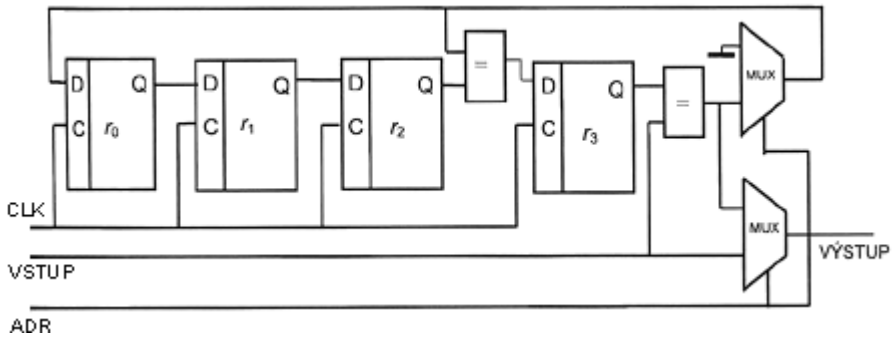
**1)** Do kodéru vstupuje *k* informačních bitů v pořadí od nejvyššího bitu. Tytéž bity z kodéru zároveň vystupují. To znamená, že spínač *S2* je v poloze 2. Spínač *S1* je po dobu přijímání informačních bitů sepnut. Na základě tohoto procesu se data dostávají do posuvných registrů. Tento děj probíhá v cyklu od 1 do *k*.

**2)** Po průchodu všech *k* informačních bitů zůstane v pamětech registru uchován poslední stav, který vyjadřuje zbytek dělení *r(x)*.

**3)** V cyklu od *k*+1 do *n* je vyprazdňován posuvný registr a všechna data jsou přenášena na výstup kodéru. Spínač *S1* je rozepnut a spínač *S2* je v poloze 1.

**4)** Celkový počet posunutí v posuvných registrech je *n*. Po uplynutí doby potřebné k zakódování se cyklus opakuje s novou posloupností informačních bitů.

Na obrázku (Obr. 7) je znázorněn obvod, který využívá děličku mod*g(x)* s tzv. přednásobením $x^{n-k}$. V takovém případě vstupuje posloupnost informačních bitů v nezměněné podobě do děličky mod*g(x)* přes sčítačku mod*2*, do které vstupuje také zpětnovazební signál, vystupující z poslední paměťové buňky děličky, tj. paměťové buňky $r_{n-k-1}$ . Je možná též varianta, kdy je sčítačka mod*2* se vstupem informačních bitů zapojena před první buňkou $r_0$. V takovém případě je nutné přivádět posloupnost informačních bitů doplněných o $n-k$ nulových míst. Toho dosáhneme vynásobením informačního mnohočlenu *u(x)* členem $x^{n-k}$, jak bylo uvedeno v části 2.4.2.



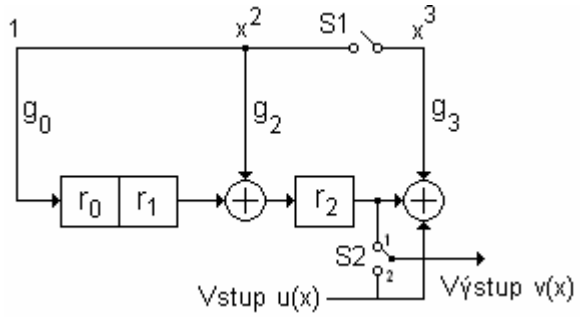

*Obr. 8. Kodér systematického (15, 11)-kódu s g(x)=* $x^4 + x^3 + 1$ [1]

Praktická realizace posuvného registru spočívá ve vzájemném propojení klopných obvodů, mezi kterými se logická informace posouvá pomocí hodinových pulsů. V případě kruhového registru je výstup spojen se vstupem. To znamená, že když logická informace

dosáhne konce registru, začne znovu obíhat kruhovým registrem. Na obrázku (Obr. 8) je zobrazen kodér s kruhovým posuvným registrem sestaveným pomocí *D klopných obvodů*, z nichž každý realizuje jednobitovou paměť.

***Příklad:***

Je dán kód (7, 4), kde informační slovo *1110*, určené k zakódování, odpovídá mnohočlenu $u(x)= x^3 + x^2 + x$. Generující mnohočlen kódu je zadán jako $g(x)= x^3 + x^2 + 1$, což odpovídá binárnímu vyjádření *1101*. Kodér je zobrazen na obrázku (Obr. 9). Kódování je popsáno pomocí tabulky (Tab. 9).




*Obr. 9. Kodér systematického kódu s* $g(x)= x^3 + x^2 + 1$

*Tab. 9. Průběh kódování*

| posun | Vstup | $r_0$ | $r_1$ | $r_2$ | S2 | S1 | Výstup |
|---|---|---|---|---|---|---|---|
| - | - | 0 | 0 | 0 | 2 | on | - |
| 1 | 1 | 1 | 0 | 1 | 2 | on | 1 |
| 2 | 1 | 0 | 1 | 0 | 2 | on | 11 |
| 3 | 1 | 1 | 0 | 0 | 2 | on | 111 |
| 4 | 0 | 0 | 1 | 0 | 2 | on | 0111 |
| 5 | - | 0 | 0 | 1 | 1 | off | 00111 |
| 6 | - | 0 | 0 | 0 | 1 | off | 100111 |
| 7 | - | 0 | 0 | 0 | 1 | off | 0100111 |

Zpráva je tedy odeslána ve tvaru : | CRC | Informační část | = | 010 | 0111 |

Při prvních *k* posuvech je odvysíláno *k* informačních bitů (od nejvyššího bitu) a obvod pro dělení mnohočlenem *g(x)* vytváří zbytek po dělení. V dalších $n-k$ krocích je pak odvysílán zbytek po dělení (zabezpečující bity), přičemž je *S1* rozpojen, aby byla na vstupu kodéru nulová hodnota a zbytek po dělení nebyl ovlivněn.

### 2.6.2 Meggitův dekodér

Pro výpočet syndromu *s(x)* přijatého slova je potřebné provést operaci dělení generujícím mnohočlenem *g(x).* Pro dekódování by tedy bylo možné opět použít děličku mod*g(x).* V tomto případě by však bylo nutné přijaté bity přivádět před první paměťovou buňku (dělička bez přednásobení), neboť mnohočlen přijatého slova je stupně $n$ a jeví se tak jako informační mnohočlen, vynásobený členem $x^{n-k}$.

Opravu chyby je možné zajistit použitím tzv. *Meggitova dekodéru*, který děličku mod*g(x)*, doplňuje o posuvný registr (*buffer*) s $n$ paměťovými buňkami a logický obvod, který zajišťuje tvorbu opravného signálu.

Uvažujeme-li přijaté slovo *w(x)*, které má syndrom *s(x)*, pak pro cyklický posuv *w(x)*, označený jako $w^{(1)}(x)$, obdržíme syndrom označený jako $s^{(1)}(x)$. Tento syndrom odpovídá zbytku po dělení mnohočlenu $xs(x)$ generujícím mnohočlenem *g(x).* Lze dokázat, že zbytky po dělení mnohočlenů $w^{(1)}(x)$ a $xs(x)$ generujícím mnohočlenem *g(x)* jsou stejné. Této skutečnosti je využito v Meggitově dekódovacím algoritmu, na kterém je založena funkce dekodéru. Postup při dekódování bude vysvětlen v následujícím příkladu.

***Příklad:***

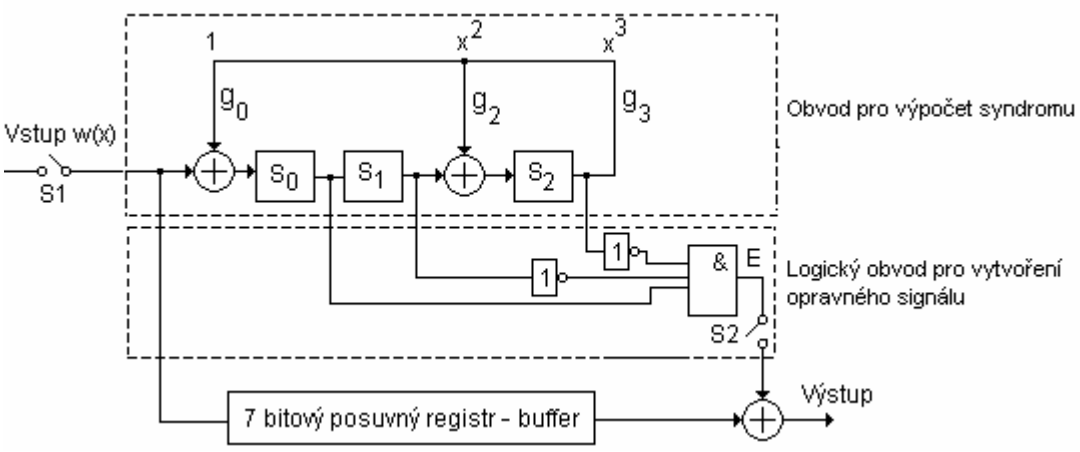
Je dán kód (7, 4), s generujícím mnohočlenem $g(x)= x^3 + x^2 + 1$. Odesláno bylo slovo *1110010*, odpovídající mnohočlenu $v(x)= x^6 + x^5 + x^4 + x$. Přijato bylo slovo *1010010*, které odpovídá mnohočlenu $w(x)= x^6 + x^4 + x$. Dekódování probíhá následovně:

**1)** Do kodéru vstupuje $n$ bitů přijatého slova v pořadí od nejvyššího bitu. Zároveň s tím je přijaté slovo ukládáno do $n$-bitového posuvného registru. Po $n$ posuvech je tedy v registru děličky syndrom *s(x)* přijatého slova *w(x)* a v bufferu je uloženo přijaté slovo *w(x).*

**2)** Na vstup dekodéru není přiváděna žádná další informace a k výstupu bufferu je připojen logický obvod pro tvorbu opravného signálu. Uvažujeme-li přijaté slovo, které odpovídá chybovému vektoru s chybou na nejvyšší pozici $x^6$, tedy *10000000*, odpovídá syndrom tohoto slova (po $n$ krocích) podle tabulky (Tab. 8) hodnotě *110*. Posunem toho syndromu v obvodu děličky vznikne syndrom *001*, který odpovídá mnohočlenu $x^0$. Logický obvod pro tvorbu opravného signálu tedy musí být zapojen tak, aby byla jeho výstupní hodnot $E$=1, v případě, že syndrom obsažený

v registru děličky odpovídá syndromu *001*. Pozn. - jelikož jsou v zapojení dekodéru (Obr. 10) buňky pro výpočet syndromu řazeny z leva od nejnižšího řádu, je logický obvod sestaven tak, aby generoval hodnotu *E=1* v případě, že hodnoty v buňkách děličky odpovídají bitům $s_0 = 1, s_1 = 0$ a $s_2 = 0$.

**3)** V dalších *i* krocích následuje cyklický posuv $s^{(i)}$ syndromu přijatého slova a zároveň se posouvá hodnota *w* v bufferu. V případě, že prvky snydromu $s^{(i)}$ odpovídají bitům $s_0 = 1, s_1 = 0$ *a* $s_2 = 0$, výstupem logického obvodu bude hodnota *E*=1 a bit, který je na výstupu bufferu, bude opraven. Následující tabulka (Tab. 10) popisuje průběh dekódování. Uvažujeme, že v děličce je nyní uložen syndrom přijatého slova a v bufferu je uloženo přijaté slovo.




*Obr. 10. Meggitův dekodér pro kód (7, 4) s g(x)=* $x^3 + x^2 + 1$

*Tab. 10. Průběh dekódování v Meggitově dekodéru s g(x)=* $x^3 + x^2 + 1$

| $s^{(i)} = s_0 \quad s_1 \quad s_2$ | Opravovaný bit $\oplus$ E | Výstup |
|---|---|---|
| $s^{(1)} = 0 \quad 1 \quad 1$ | $1 \oplus 0$ | 1 |
| $s^{(2)} = 1 \quad 0 \quad 0$ | $0 \oplus 1$ | 11 |
| $s^{(3)} = 0 \quad 1 \quad 0$ | $1 \oplus 0$ | 111 |
| $s^{(4)} = 0 \quad 0 \quad 1$ | $0 \oplus 0$ | 0111 |
| $s^{(5)} = 1 \quad 0 \quad 1$ | $0 \oplus 0$ | 00111 |
| $s^{(6)} = 1 \quad 1 \quad 1$ | $1 \oplus 0$ | 100111 |
| $s^{(7)} = 1 \quad 1 \quad 0$ | $0 \oplus 0$ | 0100111 |

Zapíšeme-li výstupní slovo v opačném pořadí, s nejvyšším bitem vlevo, odpovídá vektor dekódovaného slova *1110010* vektoru slova odeslaného. Oprava chyby na páté pozici tedy proběhla správně. Následující algoritmus, popisující hledání chyby, je uveden v lit. [5].

```
begin
shift received sequence into syndrome circuit;
      if syndrome zero then no errors
      else
            begin
            j: = n - 1;
            while syndrome <> 10...0 and j > 0 do
                  begin
                  j: = j – 1;
                  shift syndrome register;
                  end;
            if syndrome = 10...0 then error in bit j
            else uncorrectable error;
            end;
end.
```

Podrobnější vysvětlení principů Meggitova dekodéru je uvedeno v literatuře [1] a [5].

## 2.7 Použití cyklických kódů

Díky úspornosti a jednoduchosti implementace kódového systému jsou cyklické kódy velice rozšířené. Hlavní uplatnění nacházejí především při přenášení dat v počítačových sítích a při ukládání, čtení a skladování dat v pamětech. Při síťových přenosech bývají využívány zejména pro jejich schopnost detekce chyb. Při přenosu zprávy doplněné o kontrolní informace vypočítaná na straně odesílatele slouží k ověření korektnosti zprávy na straně příjemce, který může požádat o její přeposlání. Některé z rozšířených typů cyklických kódů, uvedených v lit. [6], jsou zobrazeny v tabulce (Tab. 11).

*Tab. 11. Rozšířené typy CRC*

| Počet kontrolních bitů | Označení | Generující mnohočlen | Použití |
|---|---|---|---|
| 8 | LRCC-8 | $x^8 + 1$ | Kontrolní součet Byte |
| 8 | CRC-8-CCITT | $x^8 + x^7 + x^3 + x^2 + 1$ | datové sběrnice |
| 12 | CRC-12 | $x^{12} + x^{11} + x^3 + x^2 + x + 1$ | Zabezpečení 6-ti bitových znaků |
| 16 | LRCC-16 | $X^{16} + 1$ | Kontrolní součet Word |
| 16 | CRC-16-IBM | $x^{16} + x^{15} + x^2 + 1$ | Synchronní protokol SDLC, USB, XMODEM |
| 16 | CRC-16-CCITT | $x^{16} + x^{12} + x^5 + 1$ | Bluetooth, IrDA |
| 32 | CRC-32 | $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ | IEEE 802.3 Ethernet, MPEG-2, RAR, PKZip |
| 64 | CRC-64 | $x^{64} + x^4 + x^3 + x + 1$ | HDLC — ISO 3309 |

# II. PRAKTICKÁ ČÁST

# 3 TECHNOLOGIE POUŽITÉ V PROGRAMU

V této části je stručný popis vývojových nástrojů a technologií, které byly použity při tvorbě programu *CyklickeKody*.

## 3.1 Programovací jazyk Java

*Java* je objektově orientovaný programovací jazyk, který vyvinula firma *Sun Microsystems*. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech (čipové karty, mobilní telefony, desktopové počítače, rozsáhle distribuované systémy). Tyto technologie se jako celek nazývají *platforma Java*. Od 8.května 2007, kdy *Sun* uvolnil zdrojové kódy, je *Java* dále vyvíjena jako open source.

*Java Runtime Environment (JRE)* je prostředí pro běh programů v *Javě*, které zahrnuje interpret *Javy* a standardní knihovny. *Java Development Kit (JDK)* obsahuje *JRE* plus překladač a další vývojové nástroje. V současné době je aktuální verze *JDK 1.6*, která byla využita také při tvorbě programu *CyklickeKody*.

Základní vlastnosti jazyka *Java* [8]:

- jednoduchý – syntaxe je zjednodušenou (a drobně upravenou) verzí syntaxe jazyka *C* a *C++*.
- objektově orientovaný – s výjimkou osmi primitivních datových typů jsou všechny ostatní datové typy objektové.
- interpretovaný – místo skutečného strojového kódu se vytváří pouze tzv. mezikód (bajtkód). Tento formát je nezávislý na architektuře počítače, operačním systému nebo typu zařízení. Program tedy pracuje na libovolném počítači nebo zařízení, které má k dispozici interpret *Javy*, tzv. *virtuální stroj Javy - Java Vitrual Machine*. Z tohoto však plyne nevýhoda, že start programů psaných v *Javě* je pomalejší, protože prostředí musí program nejprve přeložit a potom teprve spustit.
- výkonný – přestože se jedná o jazyk interpretovaný, není ztráta výkonu významná, neboť překladače pracují v režimu „*právě včas*“ a do strojového kódu se překládá jen ten kód, který je opravdu zapotřebí.
- dynamický – *Java* byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.

### 3.1.1 Java Web Start

*Java Web Start* je technologie určená ke snadné distribuci a spouštění grafických aplikací v *Javě* prostřednictvím internetu nebo intranetu. Základní princip této technologie spočívá ve vytvoření souboru ve formátu nazvaném *JNLP* (*Java Network Launching Protocol*), který obsahuje popis aplikace a informace nutné ke spuštění. Na webu je vystaven odkaz na *JNLP* soubor, který je u klienta asociován s programem *javaws*. Po kliknutí na odkaz je tento nástroj spuštěn, z daného souboru přečte potřebné informace, na jejichž základě stáhne potřebné *jar* archívy s knihovnami i vlastním programem a požadovaný program spustí. Jediným požadavkem je, aby na počítači klienta byla nainstalována *Java* s podporou *Java Web Start* (ta je standardní součástí od verze *1.4*).

### 3.1.2 Javadoc

*Javadoc* je softwarová pomůcka (utilita) firmy *Sun Microsystems*, která slouží pro automatické generování dokumentace k programu. Jde o malou konzolovou aplikaci, která prochází zdrojový kód programů v jazyku Java, vyhledává speciálně uvozené komentáře a ty posílá na výstup současně generovaného *HTML* souboru.

## 3.2 Eclipse

*Eclipse* je open source vývojová platforma, která je pro většinu lidí známa jako vývojové prostředí *(IDE)* určené pro programování v jazyce *Java*. V základní verzi obsahuje pouze integrované prostředky jako kompilátor, debugger atd. Ostatní vývojové prostředky lze dodávat prostřednictvím pluginů.

### 3.2.1 Eclipse visual editor

*Visual editor* je volně dostupný plugin do prostředí *Eclipse*, jehož prostřednictvím lze vytvářet grafické uživatelské rozhraní. Zdrojový kód GUI je vytvářen automaticky přímo během návrhu ve vizuálním módu. Styl generování kódu bývá označován také jako *getterStyle*. Grafické komponenty jsou zde vytvořeny jako globální proměnné, přičemž je téměř pro každou komponentu vytvořena vlastní metoda (*getter*), která obsahuje nastavení a vlastnosti této komponenty.

# 4 PROGRAM PRO TVORBU CYKLICKÉHO KÓDU

Pro ověření principů pospaných v teoretické části byl v jazyce *Java* vytvořen program s názvem *CyklickeKody*. Jeho hlavní funkce spočívá ve vykreslení obvodu kodéru a dekodéru na základě zadaných dat a zaznamenání stavu těchto obvodů v jednotlivých fázích kódování a dekódování. Program je zaměřen na cyklické kódy pro opravu jedné chyby.

## 4.1 Struktura programu

Při tvorbě projektů v jazyce *Java* je využíváno tzv. balíků (*packages*), které slouží k vytváření nových prostorů jmen (*namespaces*) a umožňují tak například vytváření knihoven, tříd a rozhraní. Každý balík je reprezentován adresářem, který obsahuje zdrojové kódy jednotlivých tříd (soubory *.java*) a přeložené třídy (soubory *.class*). Symboly používané v prostředí Eclipse jsou následující:

- *Balík (package),* *Soubor se zdrojovým kódem,* *Třída (class)*
- *Konstruktor dané třídy,* *Metoda (funkce) dané třídy.*

Vnitřní struktura programu (projektu) *CyklickeKody* je zachycena na obrázku (Obr. 11).


*Obr. 11. Vnitřní struktura programu*

## 4.2 Popis tříd a metod programu

V této části budou stručné pospány jednotlivé třídy programu a nejdůležitější metody, které tvoří jádro programu.

### 4.2.1 Třída CyklickeKody

Definice : *public class **CyklickeKody** extends JFrame*

Tato třída obsahuje metodu ***main,*** kde je vytvořena nová instance třídy a z konstruktoru je pak volána metoda ***initialize***. Prostřednictvím této metody jsou zavolány metody (*gettery*) jednotlivých grafických komponent, čímž dojde k inicializaci grafického rozhraní programu. Třída je vytvořena v *Eclipse visual editoru* jako tzv. *Visual Class.* Z větší části tedy obsahuje metody, které byly vytvořeny automaticky pro každou komponentu. Tyto metody jsou hierarchicky propojeny a jejich detailní popis by byl velice zdlouhavý. Důležité je však uvést, že komponenty jako tlačítka nebo textová pole mají definovánu vnitřní metodu, která zajišťuje obsluhu události jako je např. kliknutí myší či stisknutí klávesy. Prostřednictvím těchto metod jsou volány metody jiných tříd a nastavovány vlastnosti některých komponent, jako např. deaktivace tlačítka či výpis do textového pole. Grafické prostředí je vytvořeno z větší části komponentami z grafické knihovny *Swing*.

### 4.2.2 Třída Kodovani

Definice : *public class **Kodovani***

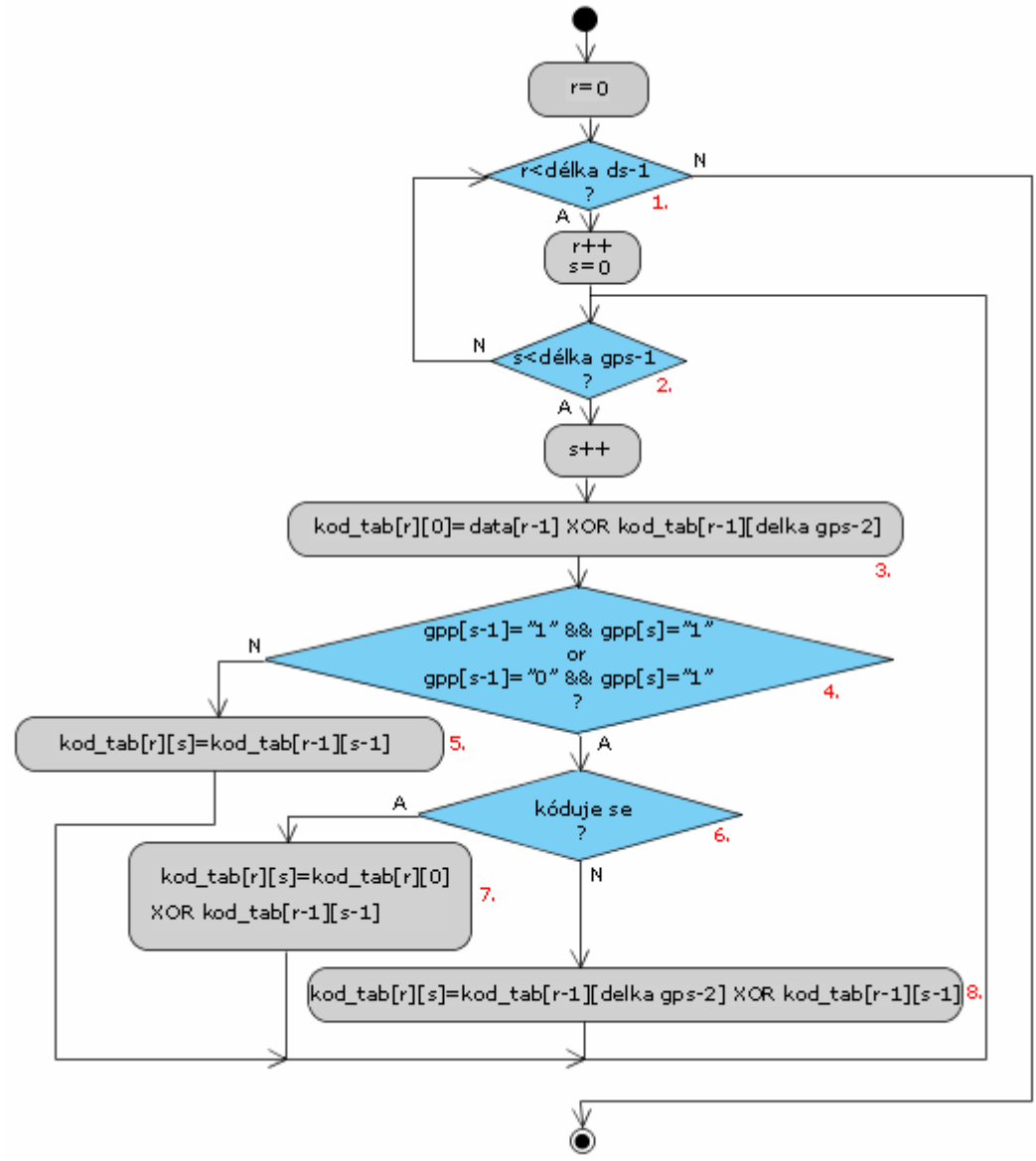
Metody v této třídě zajišťují zakódování zadaných informačních bitů a předávání výsledných hodnot jiným metodám či třídám.

- **Metoda Koduj**

Definice : *public String[][] **koduj**( String data_s, String gen_pol_s)*

Úkolem této metody je vytvoření matice ***kod_tab*** , která obsahuje hodnoty “1“ nebo “0“, popisující průběh kódování či dekódování. Jednotlivé části této matice jsou vypisovány do tabulky v okně programu. Prvky na řádcích této matice jsou vykreslovány rovněž v buňkách obvodu kodéru nebo dekodéru. Parametry metody jsou řetězce ***data_s*** a ***gen_pol_s***, které představují zadané bity informačního a generujícího mnohočlenu. Na začátku metody jsou z těchto řetězců vytvořena pole, přičemž pole odpovídající generujícímu mnohočlenu obsahuje prvky v opačném pořadí než v řetězci. První řádek matice ***kod_tab*** je naplněn inicializační hodnotou “0“ . Výpočet prvků na dalších řádcích

probíhá podle algoritmu zobrazeného ve vývojovém diagramu (Obr. 12). Řetězec ***gen_pol_s*** je v diagramu označen zkratkou ***gps*** a jemu odpovídající pole zkratkou ***gpp***. Řetězec ***data_s*** je v diagramu označen zkratkou ***ds*** a jemu odpovídající pole zkratkou ***data***.




*Obr. 12. Vývojový diagram jádra metody Koduj*

**1.** Cyklus For pro indexaci a procházení řádků matice. **2.** Cyklus For pro indexaci a procházení sloupců matice. **3.** První prvek každého řádku odpovídá součtu mod*2* (*XOR*) aktuálního vstupního bitu z pole ***data*** s posledním prvkem matice o řádek výš (řádek s indexem o 1 menším). V obvodu kodéru nebo dekodéru to odpovídá součtu mod*2* vstupního bitu s hodnotu v poslední buňce posuvného registru v minulém kroku.

**4**. Kombinace sousedních znaků "11" nebo "10" v generujícím mnohočlenu signalizuje sčítačku mod*2*. **5.** Pokud není podmínka 4. splněna, odpovídá hodnota prvku matice hodnotě sousedního prvku o řádek výš. **6**. Test, zda se kóduje či dekóduje (konstrukce kodéru a dekodéru není zcela totožná). **7.** Pokud se kóduje, odpovídá prvek matice součtu mod*2* prvního prvku daného řádku a sousedního prvku o řádek výš. **8.** Při dekódování odpovídá prvek matice součtu mod*2* posledního prvku daného řádku a sousedního prvku o řádek výš.

- **Metoda testGP**

Definice : *public String* ***testGP****(String data_s, String gen_pol_s)*

Parametry této metody jsou řetězce ***data_s*** a ***gen_pol_s***, které představují zadané bity informačního a generujícího mnohočlenu. Pomocí hodnoty ***N***, vypočítané podle vzorce:

$$\boldsymbol{N = délka\ data\_s + délka\ gen\_pol\_s - 1} \qquad (19)$$

je vytvořen řetězec, který tvoří znak "1", dále ***N***-2 znaků "0" a na poslední pozici znak "1". Tento řetězec odpovídá mnohočlenu $x^n - 1$ pro daný kód délky *n* a je použit jako jeden parametr pro volanou metodu ***Deleni.*** Druhým parametrem je řetězec s generujícím mnohočlenem. Návratová hodnota metody ***Deleni*** odpovídá zbytku po dělení mnohočlenu $x^n - 1$ generujícím mnohočlenem. Pokud je tento zbytek nulový, je návratová hodnota metody ***testGP*** prázdný řetězec, v opačném případě vrací řetězec s upozorněním na neplatný generující mnohočlen.

- **Metoda errorVector**

Definice : *public void* ***errorVector****(String data_s, String gen_pol_s)*

Parametry této metody jsou obdobné jako u metody ***testGP***. Podle délky zakódovaného zabezpečeného slova jsou postupně vytvořeny řetězce, reprezentující jednotlivé chybové vektory pro dané slovo. Např. pro slovo délky *7* bitů jsou postupně generovány řetězce: "0000001", "0000010", "0000100", atd. Každý takový řetězec slouží jako parametr pro volané metody ***Deleni.*** Druhým parametrem je řetězec s generujícím mnohočlenem. Návratová hodnota metody ***Deleni*** určuje syndrom daného chybového vektoru. Z hodnot syndromů je sestavena tabulka ***errorTab***. Podle toho, který syndrom v tabulce odpovídá syndromu přijatého slova vypočítaného v metodě ***Koduj***, je určena pozice chybného bitu v přijatém slově.

- **Metoda Deleni**

Definice : *public String* ***Deleni****(String delit, String delen)*

Tato metoda zjišťuje výpočet zbytku po dělení dvou binárních čísel, které jsou předány metodě jako parametry. Parametr ***delit*** představuje dělitele, parametr ***delen*** představuje dělenec. Pro výpočet zbytku po dělení je využit postup, který je uveden na následujícím schématu. Princip je podobný, jako v příkladech uvedených v kapitolách 2.4.2 a 2.5.

| | | |
|---|---|---|
| | 1 0 1 0 0 1 0 | *delenec* |
| ⊕ | 1 1 0 1 | *delitel* |
| | 1 1 1 0 1 0 | *mezikrok* |
| ⊕ | 1 1 0 1 | *delitel* |
| | 1 1 10 | *mezikrok* |
| ⊕ | 1 1 01 | *delitel* |
| | 11 | *zbytek po deleni* |

Základem je cyklus while, který probíhá tak dlouho, dokud je délka řetězce v mezikroku větší nebo rovna délce řetězce dělitele. V tomto cyklu jsou nejprve z řetězců ***delit*** a ***delen*** vytvořena pole ***delitel*** a ***delenec***. V cyklu for je provedena operace:

$$\textbf{\textit{delenec[i]}} = \textbf{\textit{delitel[i]}}\ XOR\ \textbf{\textit{delenec[i]}} \qquad (20)$$

pro ***i*** *= 0* až délka řetězce ***delit***-1. Takto dojde k přepsání potřebného počtu hodnot v poli ***delenec***, které je pak převedeno zpět na řetězec ***delen***. V tomto nově vzniklém řetězci jsou odstraněny případné nuly na začátku, a je opakován test hlavního cyklu while. Po proběhnutí cyklu while je v poli ***delenec*** zbytek po dělení. Obsah pole ***delenec*** je převeden na řetězec, který slouží jako návratová hodnota metody ***Deleni.***

### 4.2.3 Třída Paint

Definice : *public class* ***Paint*** *extends JComponent*

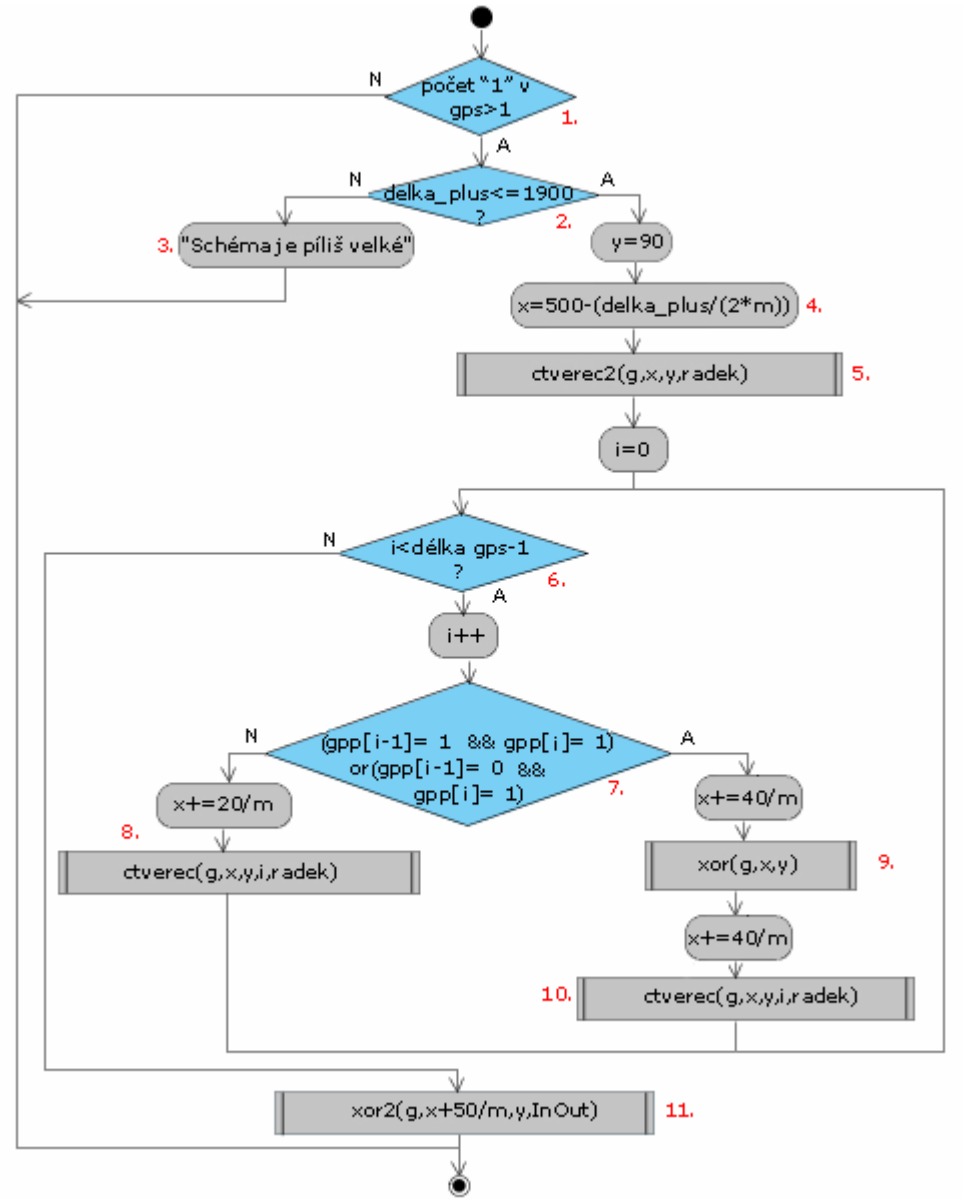
Metody této třídy zajišťují vykreslení obvodu kodéru, dekodéru, stavu na vstupu a výstupu kodéru či dekodéru a stavu v jednotlivých buňkách posuvného registru.

V metodě ***Nastav***, která zde nebude podrobněji popisována, je do globální proměnné ***delka_plus*** uložena hodnota, která je vypočítána na základě zadaného generujícího mnohočlenu a je úměrná velikosti vykreslovaného obvodu. Pokud ***delka_plus*** obsahuje hodnotu větší než 900, je do globální proměnné ***m***, jejíž význam bude vysvětlen dále, uložena hodnota 2, v opačném případě je její hodnota 1.

- **Metoda paintComponent**

Definice : *public void **paintComponent**(Graphics g)*

Tato metoda není volána přímo, ale z metody ***paint**(Graphics g)*, která je metodou zděděné třídy ***JComponent*** knihovny *Swing*. Tuto metodu grafické rozhraní volá k překreslení komponenty. Hlavním účelem této metody je volání a předávání správných parametrů jiným dílčím metodám, které zajišťují samotné vykreslení obvodu. Následující vývojový diagram (Obr. 13) popisuje funkci metody. Řetězec s generujícím mnohočlenem je označen jako ***gps***, pole s generujícím mnohočlenem jako ***gpp***.




*Obr. 13. Vývojový diagram jádra metody paintComponent*

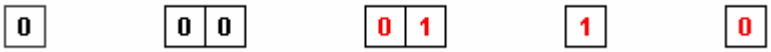
**1.** Test, zda zadaný generující mnohočlen obsahuje více než jeden znak a zda se v něm vyskytují alespoň dvě jedničky. **2.** Test zda je vypočítaná hodnota ***delka_plus*** menší nebo rovna hodnotě 1900 **3.** Pokud není předchozí podmínka splněna, vykresluje se místo obvodu řetězec s upozorněním na příliš velké schéma. **4.** Je nastavena počáteční souřadnice ***y***, od které se bude vykreslovat a vypočítána souřadnice ***x***, v závislosti na hodnotě ***m***. **5**. Je volána metoda ***ctverec2***, jejíž význam bude pospán dále. **6.** Cyklus for. **7**. Kombinace sousedních hodnot 11 nebo 10 v generujícím mnohočlenu signalizuje sčítačku *modulo 2.* **8.** Pokud není podmínka v bodu 7. splněna, je souřadnice ***x*** zvětšena o 20 a volá se pouze metoda ***ctverec***. **9.** Pokud je podmínka 7. splněna je souřadnice ***x*** zvětšena o 40, zavolá se metoda ***xor*** a souřadnice ***x*** je opět zvětšena o 40. **10.** Volání metody ***ctverec***. **11.** Po proběhnutí cyklu for je vždy volána metoda ***xor2***.

Veškeré souřadnice a rozměry jsou v této třídě a ve všech jejich metodách děleny proměnnou ***m***. Pokud je tedy zadán takový generující mnohočlen, kdy by byl vykreslovaný obvod příliš velký, jsou všechny rozměry zmenšeny na polovinu.

- **Metoda ctverec**

Definice : *public void* ***ctverec****(Graphics g, int x, int y ,int i,int r)*

Na předané parametry souřadnic ***x*** a ***y*** je vykreslován čtverec. Znak, který se vykresluje ve čtverci je dán parametry ***i*** a ***r***, které určují prvek v tabulce ***kod_tab***. Ukázka části vykresleného schématu, po (několikanásobném) volání metody ***ctverec***, je na obrázku (Obr. 14).

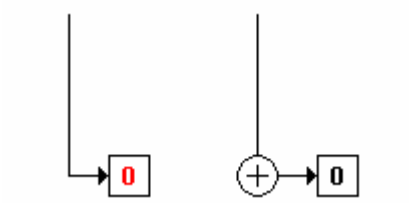



*Obr. 14. Vykreslená část obvodu I.*

- **Metoda ctverec2**

Definice : *public void* ***ctverec2****(Graphics g, int x, int y, int r)*

Na předané parametry souřadnic ***x*** a ***y*** je vykreslen první čtverec (buňka $r_0, s_0$) a část obvodu, která se nachází vlevo. Znak, který je vykreslený ve čtverci, je dán parametry 0 a ***r***, které určují prvek v tabulce ***kod_tab***. Ukázka části vykresleného schématu (vlevo pro kodér, vpravo pro dekodér) po volání metody ***ctverec*** je na obrázku (Obr. 15).

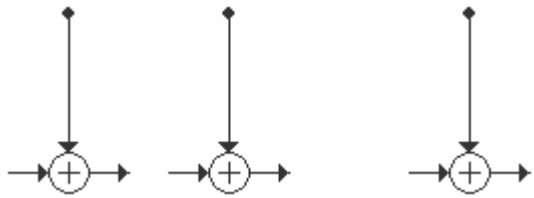



*Obr. 15. Vykreslená část obvodu II.*

- **Metoda xor**

Definice : *public void* ***xor****(Graphics g, int x, int y)*

Na předané parametry souřadnic ***x*** a ***y*** je vykreslena sčítačka *modulo 2*, doplněná o šipky představující vstupy a výstupy sčítačky. Ukázka části vykresleného schématu po (několikanásobném) volání metody ***xor*** je na obrázku (Obr. 16).

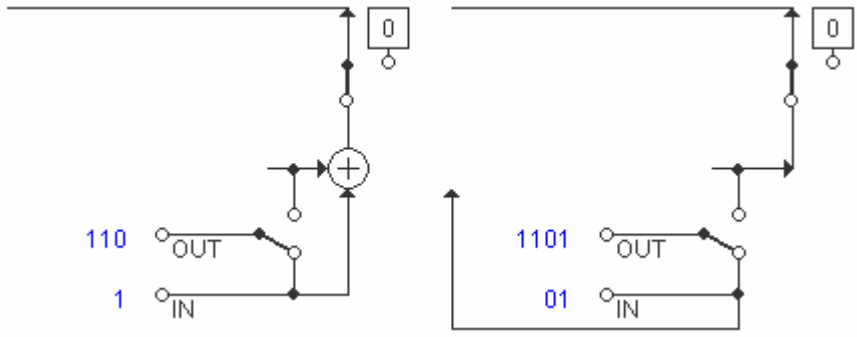


*Obr. 16. Vykreslená část obvodu III.*

- **Metoda xor2**

Definice : *public void* ***xor2*** *(Graphics g,int x,int y,String[] dataIO)*

Na předané parametry souřadnic ***x*** a ***y*** je vykreslena zbývající část obvodu, tedy sčítačka *modulo 2* nejvíce vpravo (kodér), vlevo (dekodér), vstup IN, výstup OUT a přepínače. Je zde též zajištěno vykreslení aktuálního vstupního řetězce u vstupu IN a výstupu OUT. Ukázka části vykresleného schématu (vlevo pro kodér, vpravo pro dekodér) po volání metody ***xor* 2** je na obrázku (Obr. 17).




*Obr. 17. Vykreslená část obvodu IV.*

Parametr ***g*** u jednotlivých metod představuje *grafický kontext*, který reprezentuje instanci třídy ***java.awt.Graphics***. Jedná se tedy o objekt, pomocí kterého je možné provádět grafický výstup. Metody třídy ***Graphics***, které byly použity pro vykreslování jednotlivých grafických entit, jsou uvedeny v tabulce (Tab. 12).

*Tab. 12. Využívané metody třídy Graphics*

| Grafická entita | Metoda třídy Graphics |
|---|---|
| **Linie** - vykreslení spojů ve schématu. | ***drawLine****(int x1, int y1, int x2, int y2)* |
| **Elipsa** - vykreslení kružnic sčítaček. | ***drawOval****(int x, int y, int sirka, int vyska)* |
| **Obdélník** - vykreslení buněk posuvného registru. | ***drawRect****(int x, int y, int sirka, int vyska)* |
| **Řetězec** - vykreslení řetězců na vstupu IN, výstupu OUT a v jednotlivých buňkách. | ***drawString****(String text, int x, int y)* |
| **Vyplněný polygon** - šipky. | ***fillPolygon****(int [x], int [y], int pocet_vrcholu)* |

### 4.2.4 Třída Vizualizace

Definice : *public class **Vizualizace***

V této třídě se nachází metoda ***testBin***, která prochází přijatý řetězec znak po znaku a testuje, zda se v něm nachází pouze hodnoty "0" a "1". Je volána vždy během stisku klávesy v nějakém z textových polí. V případě, že testovací řetězec není tvořen binárním číslem, je návratová hodnota metody řetězec s upozorněním. Další metoda této třídy, s názvem ***Zobraz_gp***, vrací na základě přijatého řetězce s binárním číslem řetězec s polynomiálním vyjádřením tohoto čísla (slova). Metoda je volána vždy při uvolnění klávesy v textovém poli pro zadávání generujícího mnohočlenu či informačních bitů. Návratová hodnota se vypisuje do textového panelu pod daným textovým polem. Poslední metodou této třídy je metoda ***DekodInf***, která na základě přijatých dat vypíše informace o proběhlém kódování a dekódování do textového panelu na záložce *Dekódování*.

### 4.2.5 Třídy RendererTabulky

Definice : *class **RendererTabulkyKD** extends DefaultTableCellRenderer*

: *class **RendererTabulkyS** extends DefaultTableCellRenderer*

První uvedená třída zajišťuje vekreslení tabulky zobrazující průběh kódování a dekódování, druhá třída slouží pro vykreslení tabulky se syndromy jednotlivých chybových vektorů. Složitější grafické komponenty ve *Swingu*, jako je právě tabulka, jsou hierarchicky složeny

z objektů a komponent menších a jednodušších. Patří mezi ně i objekty označované jako *renderery*, které slouží k vykreslování dat obsažených v tabulce. Obě třídy popisované v této části obsahují metodu ***getTableCellRendererComponent***, ve které *renderer* vrací grafickou komponentu (odvozenou od ***java.awt.Component***) určenou pro vykreslení dané buňky tabulky. Podle indexu této buňky je rozhodnuto, jaká bude barva jejího pozadí.

### 4.2.6 Třída UpravaTabulky

Definice : *public class **UpravaTabulky** extends DefaultTableModel*

Proces mazání a znovu vytváření tabulky se ukázal v průběhu tvorby programu jako poměrně komplikovaný. Pro zpřehlednění zdrojového kódu byla tedy vytvořena tato třída, která obsahuje metodu ***removeColumn*** zajišťující odstranění sloupce tabulky určeného indexem, který je předán jako parametr.

### 4.2.7 Dokumentace Javadoc

Přehled a stručný popis všech metod, které jsou využívány tímto programem, je uveden v dokumentaci vygenerované prostřednictvím nástroje *Javadoc*. Tuto dokumentaci lze nalézt na webové stránce, která bude zmíněna dále.

## 4.3 Popis a obsluha programu

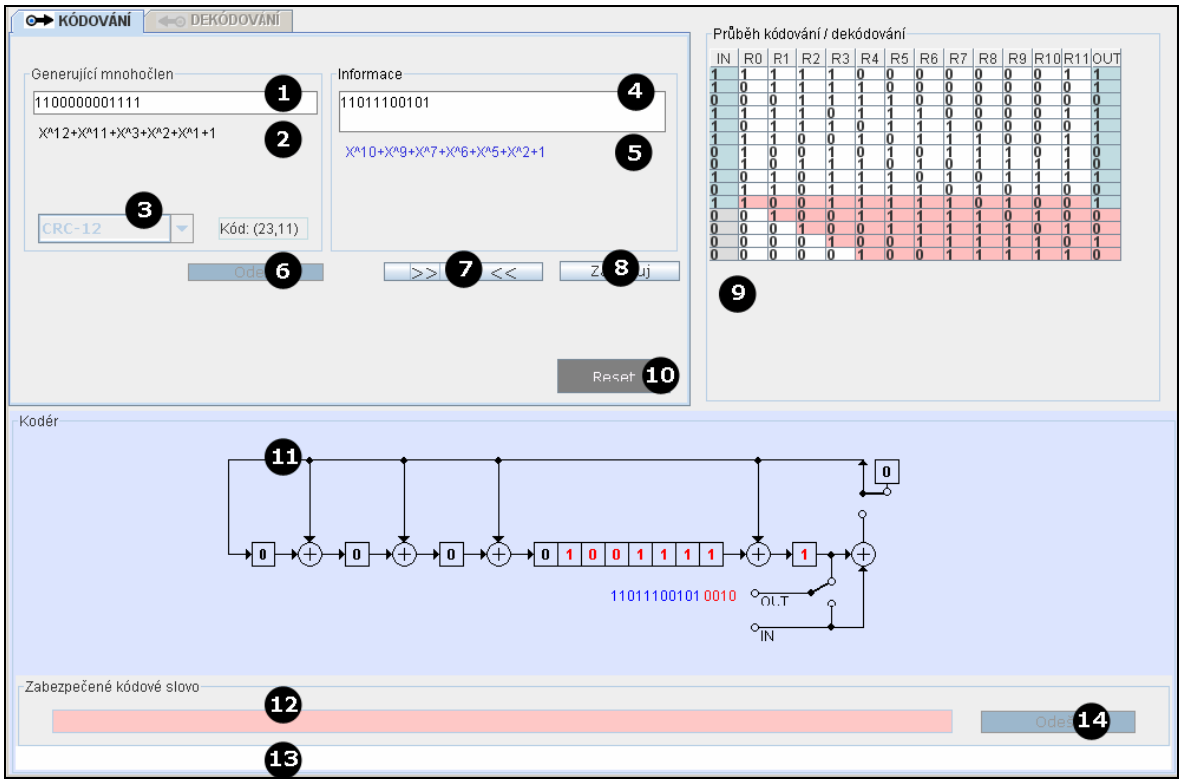
### 4.3.1 Okno programu

Okno programu *CyklickeKody* je složeno ze tří hlavních částí. První částí je plocha, kde je vykreslován obvod kodéru a dekodéru, druhou část tvoří tabulka, která se vytváří postupně během kódování a dekódování a třetí část je tvořena dvěmi záložkami, které obsahují ovládací prvky, vstupní textová pole a informační textové panely.

Okno programu s aktivní záložkou s názvem *Kódování*, je zobrazeno na obrázku (Obr. 18). Jednotlivé prvky okna označené čísly mají následující význam:

1. textové pole pro zadání generujícího mnohočlenu
2. textový panel pro zobrazení generujícího mnohočlenu
3. combo box pro výběr předvolených generujících mnohočlenů
4. textové pole pro zadání informačních bitů
5. textový panel pro zobrazení informačního mnohočlenu

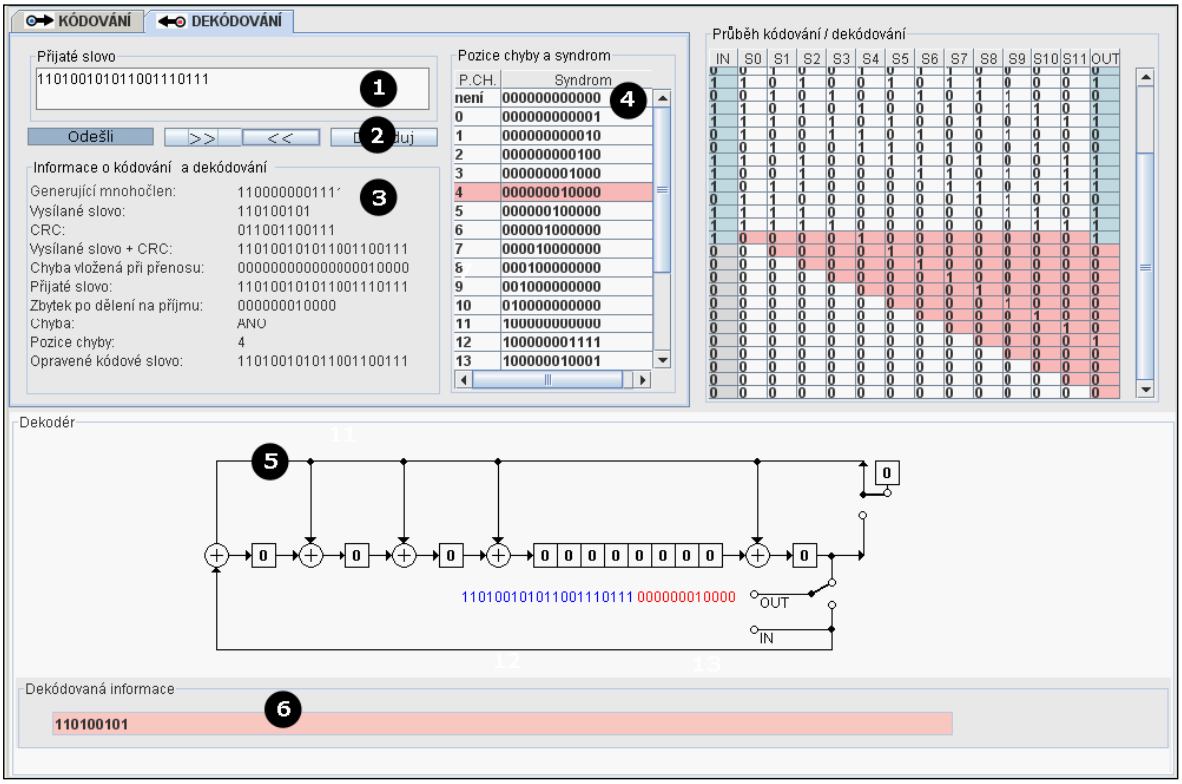
6. tlačítko *Odeslat* - odeslání zadaných informačních bitů na kodér
7. tlačítka >> << - krokování kódování: tam a zpět
8. tlačítko *Zakóduj* - provede kompletní zakódování v jednom kroku
9. tabulka zachycující průběh kódování a dekódování
10. tlačítko *Reset* - vymazání stávajících hodnot jednotlivých komponent
11. schéma kodéru odpovídající zadanému generujícímu mnohočlenu
12. textové pole pro zobrazení zabezpečeného (zakódovaného) slova
13. textový panel pro zobrazování upozornění
14. tlačítko *Odeslat* - „odeslání“ zabezpečeného slova přenosovým kanálem




*Obr. 18. Okno programu s aktivní záložkou Kódování*

Okno programu, kde je aktivní záložka s názvem *Dekódování*, je zachyceno na obrázku (Obr. 19). Označeny jsou ty prvky okna, které jsou odlišné od prvků na záložce *Kódování*. Jejich význam je následující:

1. textové pole pro zobrazení a případnou editaci přijatého slova
2. tlačítko *Dekóduj* - provede dekódování v jednom kroku (Význam ostatních tlačítek je stejný jako na záložce *Kódování*.)
3. textový panel s informacemi o proběhlém kódování a dekódování
4. tabulka zobrazující pozici chyby pro daný chybový vektor a odpovídající syndrom
5. schéma dekodéru odpovídající zadanému generujícímu mnohočlenu
6. textový panel pro zobrazení dekódovaného slova

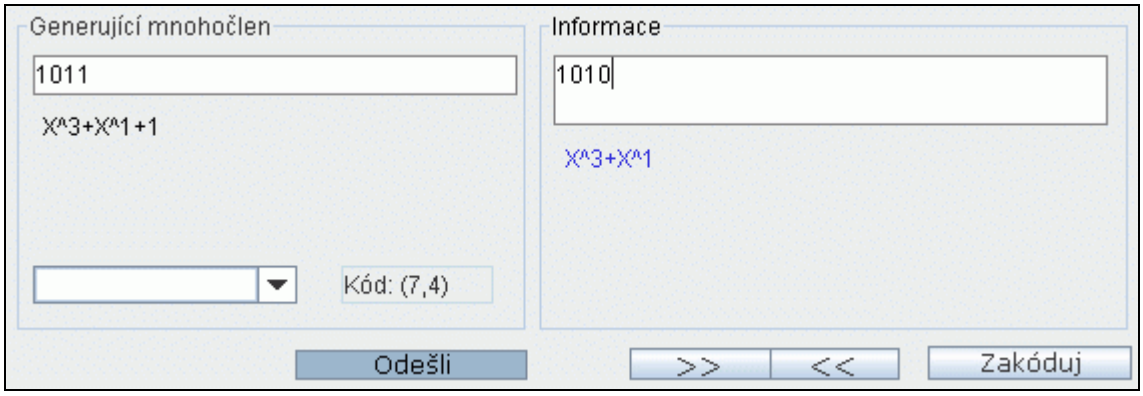



*Obr. 19. Okno programu s aktivní záložkou Dekódování*

### 4.3.2 Ovládání programu

Program *CyklickeKody* je vytvořen tak, aby bylo jeho ovládání jednoduché, a aby bylo možné v každém kroku provádět pouze ty činnosti, které jsou nezbytně nutné. Není navržen pro to, aby byl výkonným kódovacím a dekódovacím nástrojem, ale aby sloužil spíše jako průvodce procesem kódování a dekódování cyklických kódů s grafickou demonstrací. Následující popis obsluhy programu je vysvětlen na příkladu se zadaným generujícím mnohočlenem $g(x) = x^3 + x + 1$ a informačním mnohočlenem $u(x) = x^3 + x$.

- **Zadávání vstupních dat**




*Obr. 20. Část okna programu sloužící pro zadávání vstupních dat*

Aby bylo možné provádět jakékoliv další činnosti, je nejprve nutné zadat binární vstupní data, která jsou charakterizována generujícím mnohočlenem a informací určenou k zakódování. Pro zadání generujícího mnohočlenu slouží textové pole na panelu označeném jako *Generující mnohočlen*. V průběhu zadávání se pod textové pole vypisuje příslušný generující mnohočlen. Na obrázku (Obr. 20) si lze povšimnout, že zadaný bit nejvíce vlevo charakterizuje nejvyšší mocninu mnohočlenu, je tedy zvolen stejný způsob zápisu, jaký byl používaný v jednotlivých příkladech teoretické části práce. Je též možné zvolit si některý z předvolených generujících mnohočlenů prostřednictvím combo boxu, umístěném v dolní části panelu. Během zadávání je do panelu označeného jako *Kodér* vykreslován obvod kodéru (Obr. 22), který odpovídá aktuálnímu zadanému generujícímu mnohočlenu.

Informační bity lze zadávat do textového pole na panelu *Informace*. Během zdávání je opět do textového panelu pod textovým polem vypisován informační mnohočlen.

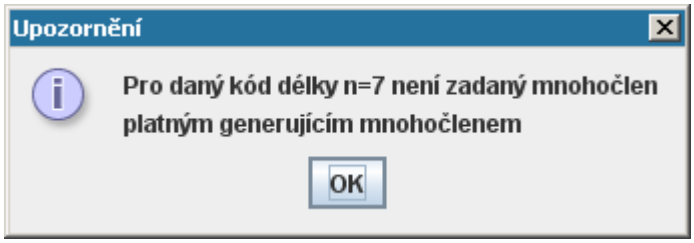
V případě zadání chybných dat je příslušné textové pole zablokováno a ve spodním informačním panelu se objeví upozornění:

| Zadaná data nejsou binární - Stiskněte prosím Back Space |

Dokud tedy není chybný znak vymazán, je nejen znemožněno zadává dalších znaků, ale je také deaktivováno tlačítko *Odešli*, kterým lze informační bity odeslat k zakódování.

Po zadání požadovaných dat ve správném tvaru je tlačítko *Odešli* aktivní, a po jeho stisknutí jsou informační bity odeslány na kodér. Po stisku je volána metoda ***testGP,*** která byla popsána v předchozí kapitole. Pokud tedy není pro daný kód délky *n* zadaný

mnohočlen platným generujícím mnohočlenem ( zbytek po dělení $\frac{x^n - 1}{g(x)}$ je nenulový), je zobrazeno informační okno s upozorněním, které zachycuje obrázek (Obr. 21).



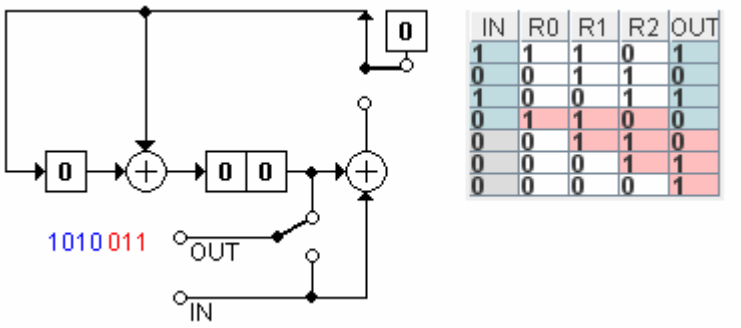

*Obr. 21. Informační okno*

- **Kódování**

V této fázi je uživateli znemožněno zadávání jakýchkoliv vstupních dat a aktivní jsou pouze tlačítka tam a zpět: <<, >> a tlačítko *Zakóduj*. Kódování lze provést krokováním tlačítkem >> (tlačítkem << se lze vracet o krok zpět), nebo tlačítkem *Zakóduj*, které provede zakódování v jednom kroku. Během kódování je v panelu *Průběh kódování / dekódování* generována tabulka, která zachycuje stav buněk a aktuální vstupní a výstupní bit na kodéru v jednotlivých krocích. Kdykoliv během kódování je možné stiskem tlačítka *Reset* vymazat stávající data a přistoupit k novému zadávání.

Barvené značení buňek tabulky a bitů v buňkách kodéru si odpovídá. Je tedy snadno rozpoznatelná původní informace (modrá barva) a zbytek po dělení (červená barva). Obrázek (Obr. 22) zachycuje stav tabulky a obvodu kodéru po proběhnutém kódování. Je možné si povšimnout, že navržený kodér vrací slovo ve tvaru:

| Informační část | CRC |
|---|---|




| IN | R0 | R1 | R2 | OUT |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 |

*Obr. 22. Vykreslený obvod kodéru a tabulka jeho stavů*

- **Odeslání zabezpečeného kódového slova**

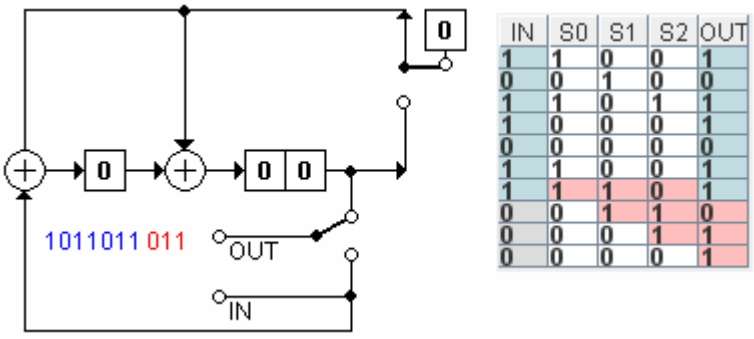
Po zakódovaní zadaných informačních bitů je v textovém poli na panelu *Zabezpečené kódové slovo* zobrazeno zabezpečené kódové slovo a je aktivováno tlačítko *Odešli*, které se nachází rovněž na tomto panelu. Po stisku tlačítka *Odešli* je aktivována záložka *Dekódování*, na kterou se je nyní možné přepnout.

- **Úprava přijatého kódového slova**

Při přepnutí okna na záložku *Dekódování* je deaktivována záložka *Kódování*. V textovém poli na panelu *Přijaté slovo* je uloženo přijaté slovo, které je možné editovat, a zavést tak na určité pozici slova chybu. Pokud přijeté slovo po případné editaci neobsahuje žádné nepovolené znaky, je možné jej tlačítkem *Odešli* odeslat ne dekodér. Celý tento krok lze provádět opakovaně (i během dekódování), což umožňuje zavádět chybu na různých pozicích.

- **Dekódování**

Stejně jako při kódování, lze provést dekódování krokováním pomocí tlačítka >> (tlačítkem << se lze vracet o krok zpět) nebo pomocí tlačítka *Dekóduj*, které provede dekódování v jednom kroku. Během dekódování je v panelu *Průběh kódování / dekódování* generována tabulka, která zachycuje stav buněk a aktuální vstupní a výstupní bit na dekodéru v jednotlivých krocích.




| IN | S0 | S1 | S2 | OUT |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 |

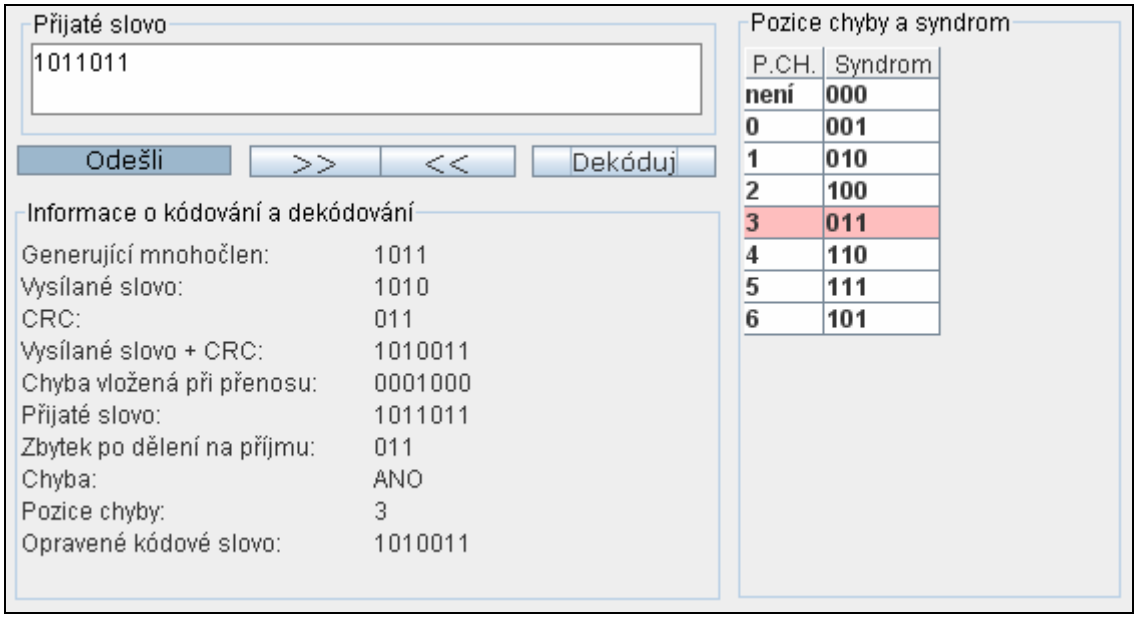
*Obr. 23. Vykreslený obvod dekodéru a tabulka jeho stavů*

Navržený obvod dekodéru, zobrazený na obrázku (Obr. 23), není schopen opravy chybného bitu, ale slouží pouze pro výpočet syndromu přijatého slova. Jeho funkce je tedy obdobná funkci kodéru, s tím rozdílem, že je využita dělička mod$g(x)$ bez přednásobení.

Stejně jako u kodéru jsou po průchodu všech bitů přivedených na vstup dekodéru přepnuty oba spínače a ke stávajícím bitům na výstupu je připojen syndrom přijatého slova. Po skončení dekódování je na výstupu dekodéru slovo ve tvaru: | Přijaté slovo | Syndrom | Takové slovo, jako celek, nemá žádný další význam, neboť v danou chvíli je důležitý především vypočítaný *syndrom*.

S posledním krokem dekódování je v panelu *Pozice chyby a syndrom* vygenerována tabulka zachycená na obrázku (Obr. 24) vpravo. Ta obsahuje v prvním sloupci čísla, která určující pozici chyby a odpovídající syndrom přijatého slova s jednou chybou na této pozici. Jsou tedy vygenerovány všechny možné syndromy, které mohly (při výskytu jedné chyby) nastat. Řádek této tabulky, který obsahuje stejný syndrom, jaký byl vypočítaný v obvodu dekodéru, je označen červenou barvou. V textovém panelu *Informace o kódování a dekódování* jsou vypsány nejdůležitější informace o proběhlém kódování a dekódování. Informační bity, které byly získány z opraveného kódového slova jsou vypsány v dolním textovém poli na panelu *Dekódovaná informace*. V případě, že se v přijatém slově vyskytnula více než jedna chyba, je zde vypsáno upozornění: **Nelze jednoznačně učit**.

Po skončení dekódování je aktivována záložka *Kódování*, kde je možné pomocí tlačítka *Reset* vymazat všechny stávající data a přistoupit k novému zadávání.




*Obr. 24. Stav okna po skončení dekódování*

## 4.4 Umístění programu na internet

Z důvodu umožnění snadné dostupnosti programu byla na serveru *Internet Centrum* (IC.cz) vytvořena jednoduchá webová stránka, kde lze stáhnout spustitelný *jar* archiv s programem, nebo program spustit prostřednictvím technologie *Java Web Start*. Umístěna zde je též dokumentace k programu vytvořená nástrojem *Javadoc* a *pdf* soubor s předchozí kapitolou této práce, která obsahuje popis a vysvětlení obsluhy programu.

**URL stránky:** http://www.cyklickekody.ic.cz/

# ZÁVĚR

Cílem této práce bylo ucelení teoretických informací o cyklických kódech a vytvoření programové aplikace, ve které by bylo možné graficky demonstrovat proces kódování a dekódování.

Cyklické kódy tvoří poměrně rozsáhlou skupinu bezpečnostních kódů, do které spadá celá řada speciálních kódů, např. BCH kódy nebo RS kódy. Tato práce není zaměřena na popis těchto jednotlivých typů, je však snaha o komplexnější a obecnější vysvětlení problematiky cyklických kódů. Hlavní důraz je pak kladen na cyklické kódy pro opravu jedné chyby. Po krátkém úvodu do problematiky bezpečnostních a lineárních kódů je uvedena kapitola obsahující základní matematickou teorii, která je potřebná pro všechny typy cyklických kódů. Za hlavní klady této kapitoly považuji vysvětlení teorie na několika příkladech a ukázku konečného binárního tělesa *GF(8)* s osmi prvky. Další část práce se zabývá samotnou konstrukcí cyklických kódů a uvádí možnosti, jak zabezpečit skupinu informačních bitů pomocí kódování a odhalit případné chyby pomocí dekódování. Závěr teoretické části je zaměřen na popis konstrukce kodéru a dekodéru cyklického kódu. Opět je zde kladen důraz na vysvětlení každé dílčí části na příkladu.

Pro ověření principů pospaných v teoretické části byl vytvořen program s názvem *CyklickeKody*, jehož hlavní funkce spočívá ve vykreslení obvodu kodéru a dekodéru na základě zadaných dat a zaznamenání stavu těchto obvodů v jednotlivých fázích kódování a dekódování. Program je napsán v jazyce *Java* a je možné jej prostřednictvím technologie *Java Web Start* spouštět z internetu. Pro tyto účely byla vytvořena jednoduchá webová stránka, kde si je též možné prohlédnout dokumentaci k programu vytvořenou nástrojem *Javadoc*. Hlavním přínosem programu je především funkce automatického vykreslování konstrukce obvodu pro dělení mnohočlenů, který je základním stavebním kamenem kodérů a dekodéru cyklických kódů a možnost krokovat jednotlivé fáze kódování a dekódování v tomto obvodu. Tento program najde tedy uplatnění především jako učební pomůcka. Základní informace o programu, klíčových algoritmech a jeho umístění na internetu, jsou uvedeny v praktické části této práce.

Teoretická část práce i samotný program by mohli do budoucna sloužit jako živná půda pro zpracování informací o složitějších BCH kódech.

# CONCLUSION

The aim of this work was to complete theoretical information about cyclic codes and creating a programme aplication, in which it is possible to grafficaly demonstrate the process of coding and decoding.

Cyclic codes make large group of error protection codes where a large amount of special codes like BCH codes or RS codes are part of it. This work is not describing the individual types, but it is trying to be more complex and general about the explanation of the cyclic code's problematic. The main emphasis has been put on the cyclic codes for correction of one mistake. A brief introduction in to the problematic of error protection and linear codes is followed by a chapter containig the basic mathematical theory, which is needed for all types of cyclic codes. Accoarding to my opinion the main positive thing about this chapter is an explanation of the theory with the help of many different examples and an illustration of the binary finite field *GF(8)* with eight elements. The following part of the work is concentrating on the construction of the cyclic codes themselves and it is offering options how to secure a group of information bits with the help of coding and to descover potential errors with the help of decoding. The end of the theoretical part is concentrating on the discription of the encoder and decoder construction of the cyclic code. Again, there has been an emphasis put on the explanation of each part in an example.

There was a programme *CyklickeKody* created for checking principles described in the theoretical part, whose main function is in drawing the circuit of encoder and decoder based on the given data and recording the state of these circuits in each stage of coding and decoding. The programme is written in *Java* language and it is possible to run it on the internet through the *Java Web Start* technology. For this purposes was created simple web site where is also posible to find program documentation generated by the Javadoc tool. The main contribution of this programme is the option of automatic construction drawing of the circuit for dividing polynomials, which is the foundation stone of encoders and decoders of cyclic codes, and an option to record step by step each stage of encoding and decoding in this circuit. This programme will especially find its usage as a teaching aid. The basic information about the programme, key algorithms, and its placement on the internet, is mentioned in the practical part of this work.

The theoretical part of this work and the programme itself could possibly be the foundation of information processing for more complicated BCH codes.

# SEZNAM POUŽITÉ LITERATURY

[1] Vlček, K.: *Komprese a kódová zabezpečení v multimediálních komunikacích*, Praha, BEN – technická literatura, 2004, ISBN 80-86056-68-6

[2] Zelinka, I., Prokopová Z.: *Základy informatiky*. FT UTB, Zlín, 2005, ISBN 80-7318-299-8

[3] Birkhoff, G., Bartee, T., C.: *Aplikovaná algebra*, Alfa, Bratislava, 1981

[4] Jiroušek, R.: *Principy digitální komunikace*. Leda, Voznice, 2006, ISBN 80-7335-084

[5] Sweeney, P.: *Error Control Coding: From Theory to Practice*, John Wiley & Sons, 2002, ISBN 0-470-84356-X

[6] *Wikipedie: Otevřená encyklopedie:* Cyclic redundancy check [online]. c2008 [cit. 2008-04-10]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Cyclic_redundancy_check#Common_polynomials>

[7] Číka, P.: *Protichybové zbezpečení BCH kódem* [online]. Publikováno 13.03.2006 [cit. 2008-04-10]. Dostupný z WWW: <http://www.elektrorevue.cz/clanky/06015/index.html>

[8] *Wikipedie: Otevřená encyklopedie: Java* [online]. c2008 [cit. 24. 04. 2008]. Dostupný z WWW: <http://cs.wikipedia.org/w/index.php?title=Java&oldid=2478456>

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| $\oplus$ | symbol používaný pro sčítání *modulo 2* (odpovídá binární operaci *XOR*) |
| *BCH* kódy | Kódy označené podle iniciálů jejich objevitelů - *R.Bose, Ray-Chaudhuri* a *A. Hocquenghem*. |
| *CRC* | *Cyclic Redundancy Check* - Cyklický redundantní součet, způsob realizace kontrolního součtu. |
| *GUI* | *Graphical User Interface* - grafické uživatelské rozhraní programu. |
| *IDE* | *Integrated Development Environment* - vývojové prostředí, software usnadňující práci programátorů. |
| *JAR* | *Java ARchiv* - platformně nezávislý formát pro archivaci souborů. |
| *JDK* | *Java Development Kit* – balík základních nástrojů pro vývoj aplikací na platformě Java. |
| *JNLP* | *Java Network Launch Protocol* - obsahuje popis aplikace a informace nutné ke spuštění programu prostřednictvím internetu. |
| *JRE* | *Java Runtime Environment (JRE)* - prostředí pro běh programů v *Javě.* |
| mod*X* | *modulo X* – operace určující zbytek po dělení proměnnou *X* |
| *RS* kódy | *Reed-Solomonovy* kódy. |

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: OBSAH PŘILOŽENÉHO CD

Obsah jednotlivých adresářů přiloženého CD:

| | |
|---|---|
| **\Bakalarska_prace** | text této bakalářské práce ve formátu *DOC* a *PDF* |
| **\CyklickeKody** | spustitelný *JAR* soubor s programem *CyklickeKody* a další adresáře související s tímto programem |
| **\CyklickeKody\Projekt** | kompletní adresář projektu s programem *CyklickeKody*, vytvořený v prostředí *Eclipse* |
| **\CyklickeKody\Javadoc** | *HTML* soubory a adresáře s dokumentací programu vytvořené v *Javadoc* |
| **\CyklickeKody\Src** | zdrojové kódy jednotlivých tříd programu *CyklickeKody* |
| **\CyklickeKody\JavaInstal** | instalační soubor *Javy* (*JRE 1.6*) |
| **\Web** | offline verze vytvořené webové stránky |

# PŘÍLOHA P II: REALIZACE KÓDU SUDÉ PARITY POMOCÍ PROGRAMU CYKLICKEKODY

Paritní kód nabízí velmi jednoduchý způsob zabezpečení. Princip spočívá v zajištění konstantní hodnoty součtu mod*2* všech symbolů kódového slova.V případě binárního kódu sudé parity přidáváme ke kódovému slovu znak $e_s = 0$ nebo $e_s = 1$ tak, aby výsledné kódové slovo mělo sudý počet jedniček (*konst=0*)**.** Tento doplněný znak (bit) $e_s$ je označován jako paritní bit.

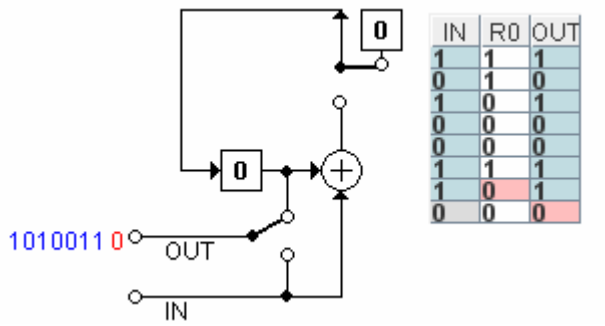
$$e_s = \sum_{i=1}^{k} \oplus u_i \quad \Rightarrow \quad e_s \oplus \sum_{i=1}^{k} \oplus u_i = konst = 0$$

Takové zabezpečení zajišťuje *minimální Hammingovu vzdálenost* kódových slov *d=2*. Pomocí paritního bitu lze detekovat lichý počet chyb (chybných bitů) ve slově, nelze však tyto chyby opravit.

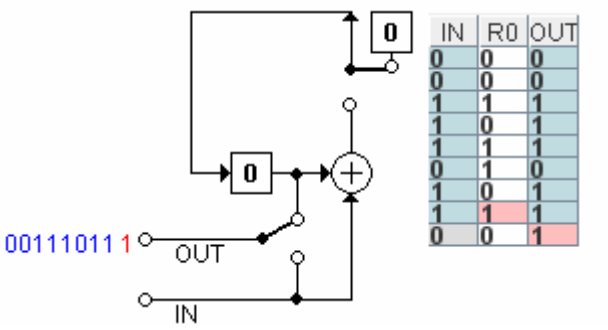
Jelikož je paritní bit speciálním případem 1-bitového CRC vypočítaného pomocí generujícím mnohočlenu *g(x)=x+1*, je možné pro jeho výpočet využít programu *CyklickeKody*.

- **Kódování**

Jako příklad jsou zvolena slova *1010011* a *00111011*. Generující mnohočlen *g(x)=x*+1, odpovídá slovu *11*. Na obrázcích (Obr. 25) a (Obr. 26) je zobrazen příslušný kodér a výsledné kódové slovo zabezpečené paritním bitem.




*Obr. 25. Zabezpečení slova I.*

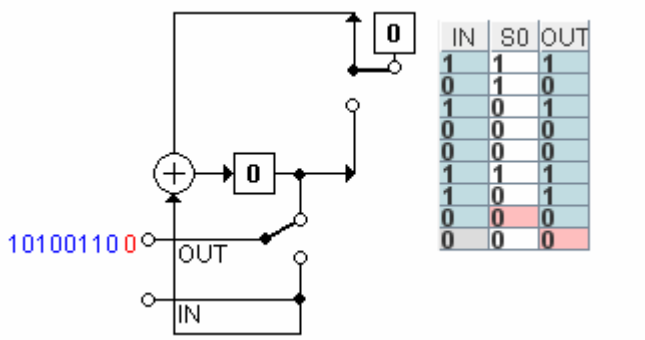



*Obr. 26. Zabezpečení slova II.*

- **Detekce chyby**

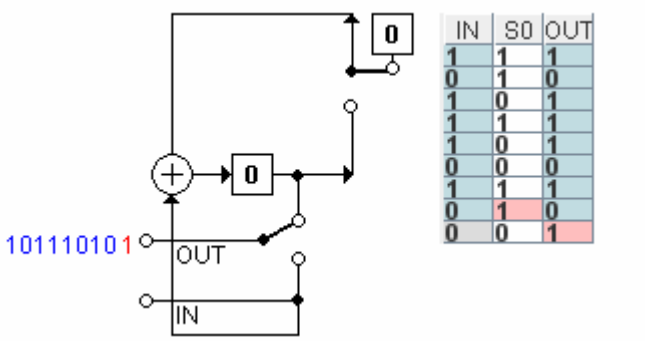
Následuje ukázka detekce chyby pro odeslané zabezpečené slovo *10100110*.

Na obrázku (Obr. 27) je zachycen dekodér a vypočítaný syndrom *0* při bezchybném přenosu (nebo při sudém počtu chyb).




*Obr. 27. Detekce chyby I.*

Na obrázku (Obr. 28) je zachycen dekodér a vypočítaný syndrom *1* na základě kterého lze detekovat, že při přenosu vznikla ve slově jednoduchá chyba (nebo lichý počet chyb). V tomto případě jsou v přijatém slově 3 chyby.




*Obr. 28. Detekce chyby II.*