Kódování – Konvoluční kódování

Coding – Convolution coding

Vít Hrbáček

Bakalářská práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Vít HRBÁČEK**
Osobní číslo:	**A07040**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Téma práce:	**Kódování – konvoluční kódování**

Zásady pro vypracování:

1. **Seznamte se s problematikou efektivních a bezpečnostních kódů, zaměřte se na konvoluční kódování.**
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 bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

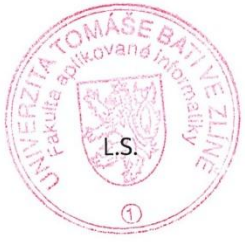
1. **Zelinka, I., Prokopová Z.: Základy informatiky. FT UTB, Zlín, 2005, ISBN 80-7318-299-8**
2. **Vlček, K.: Komprese a kódová zabezpecení v multimediálních komunikacích, Praha, BEN -- technická literatura, 2004, ISBN 80-86056-68-6**
3. **Jiroušek, R.: Principy digitální komunikace. Leda, Voznice, 2006, ISBN 80-7335-084**
4. **Kocourek,P., Novák, J.: Přenos informace, ČVUT, Paha, 2004, ISBN 8001028925**
5. **Sweeney, P.: Error Control Coding: From Theory to Practice, John Wiley & Sons, 2002, ISBN 0-470-84356-X**

Vedoucí bakalářské práce: **RNDr. Ing. Miloš Krčmář**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. března 2010**

Termín odevzdání bakalářské práce: **1. června 2010**

Ve Zlíně dne 5. března 2010



prof. Ing. Vladimír Vašek, CSc.
děkan

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

ABSTRAKT

Tato bakalářská práce se zabývá problematikou konvolučního kódování a dekódování. V teoretické části se práce zabývá potřebnými základy ke kódování obecně. Dále je podrobněji rozebráno samotné konvoluční kódování, dekódování a stavba kodéru. V praktické části je pak popsán program vytvořený pro praktickou ukázku a simulaci konvolučního kódování.

Klíčová slova: konvoluce, konvoluční kódování, dekódování, kodér, dekodér, Viterbiho algoritmus, C++

ABSTRACT

This bachelor thesis deals with the convolution encoding and decoding. The theoretical part is focused on the necessary foundations of encoding in general. The convolution encoding, decoding and encoder structure are discussed in more detail. The program created for a simulation of the convolution encoding is described in the practical part of the thesis.

Keywords: convolution, convolution encoding, decoding, encoder, decoder, Viterbi algorithm, C + +

Děkuji panu RNDr. Ing. Miloši Krčmářovi, vedoucímu bakalářské práce, za odbornou pomoc, cenné rady, připomínky a veškerý strávený čas při tvorbě této práce. Rodině za poskytnuté zázemí a podporu při studiu.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Cílem této práce je seznámení s konvolučními kódy a kódováním.

První část práce budu věnovat obecnému úvodu do kódování. Vysvětlím vztah mezi kódem, kódováním a systémem, Hammnigovu vzdálenost a její využití, rozdělení kódů a jiné. Následně podrobně rozeberu konvoluční kódy, na které je tato práce zaměřena. Ukážu základy pro konvoluční kódování, vysvětlím, z čeho se skládá kodér pro konvoluční kódování a jak sestavit kodér z generujícího polynomu. Metody použité ke kódování a dekódování konvolučního kódu. Probranou teorii také názorně ukážu na příkladech.

V druhé části práce pak popíši program, který vznikne k praktické ukázce konvolučního kódování a na kterém si vyzkouším naprogramování správného algoritmu pro kódování a dekódování konvolučního kódu. Pro kódování si zvolím algoritmus matematického modelu $v(x) = f(x) * G$. Pro dekódování pak využiji znalosti Viterbiho algoritmu a mřížkového diagramu. Pro naprogramování použiju programovací jazyk C++ a vývojového prostředí Microsoft Visual Studio 2008, které také stručně popíšu v praktické části práce.

Pro bakalářskou práci s názvem „Kódování – Konvoluční kódování“ jsem se rozhodl, protože mi toto téma bylo doposud zcela neznámé a možnost naučit se něco nového mi přišlo jako dobrý nápad. Tato práce bude moci sloužit i jako pomocný studijní materiál pro studenty, kteří také začínají s konvolučními kódy. Práce bude obohacena o spousty ukázkových příkladů.

I. TEORETICKÁ ČÁST

1 ÚVOD DO KÓDOVÁNÍ

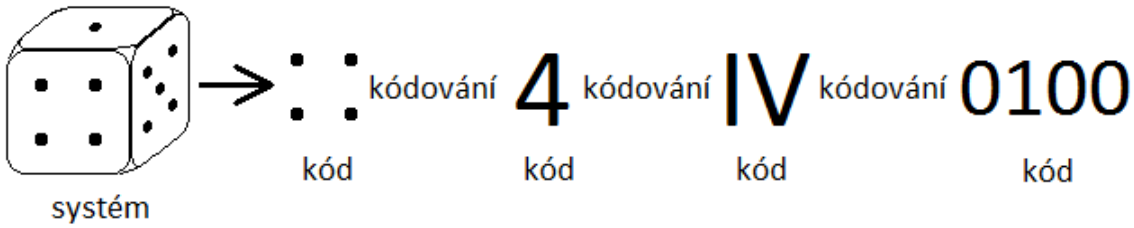
1.1 Kódy, kódování, kodér

Kód – vyjádření stavu systému. Kód může být vyjádřen různými způsoby, algebraicky, numericky, binárně, tabulkou, či jiným předem definovaným způsobem.

Kódování – provádí se při převodu mezi kódy. Zpravidla bývá prováděn kodérem.

Kodér – přístroj nebo součástka, která provádí kódování. Za kodér lze považovat třeba i lidský mozek. [4]

Vztah mezi kódem a kódováním je zobrazen na následujícím obrázku, nejčastěji se k tomu využívá hrací kostka.



Obr. 1-1. Vztah mezi kódem a kódováním

Hrací kostka je systém, který nabývá jednoho ze šesti možných stavů, tento stav je vyjádřený kódem, který je možný kódováním převést na jiný kód.

1.2 Hammingova vzdálenost

Hammingova vzdálenost – je dána počtem odlišností dvou slov A a B o stejné délce. [1], [2] a [3]

Zápis Hammingovy vzdálenosti je následující:

$$d(A,B) = \text{počet odlišností} \tag{1.1}$$

Příklad 1.1:

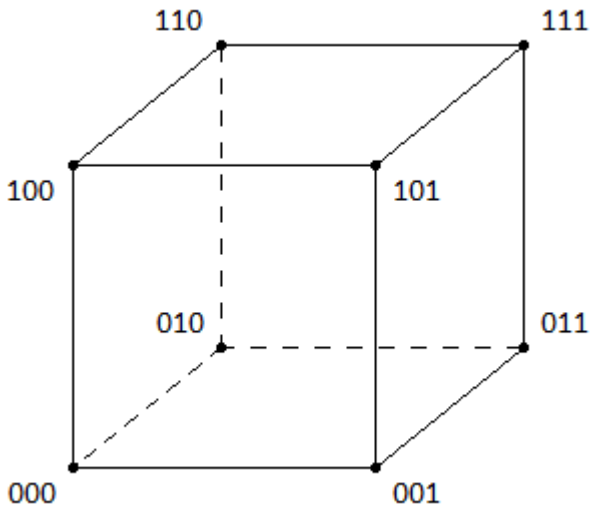
Mějme dvě slova stejné délky, A = 1010 a B = 1100. Liší se ve dvou bitech, druhém a třetím. Hammingova vzdálenost je potom:

$$d(A,B) = 2 \tag{1.2}$$

Minimální Hammingova vzdálenost – je dána nejmenším počtem odlišností dvou slov v daném kódu. Zápis minimální Hammingovy vzdálenosti je následující:

$$d_{min} = \text{nejmenší počet odlišností} \tag{1.3}$$

Pro zobrazení minimální Hammingovy vzdálenosti se nejčastěji využívá Hammingovy kostky (Obr. 1-2).



Obr. 1-2. Hammingova kosta

Na Hammingově kostce je možné ukázat tři případy pro tři různé minimální Hammingovy délky.

1) Hammingova délka má být d_{min} = 1:
   Ke slovu 000 můžeme vybrat slovo 001, 010 nebo 100. Ke slovu 100 můžeme vybrat slovo 101, 110 nebo 000, atd.

2) Hammingova délka má být d_{min} = 2:
   Ke slovu 000 můžeme vybrat slovo 011, 110 nebo 101. Ke slovu 100 můžeme vybrat slovo 111, 010 nebo 001, atd.

3) Hammingova délka má být d_{min} = 3:
   Ke slovu 000 můžeme vybrat pouze slovo 111. Ke slovu 100 můžeme vybrat pouze slovo 011, atd. Při této Hammingově délce můžeme jednu chybu opravit a při dvou chybách určit, že je přijaté slovo chybové.

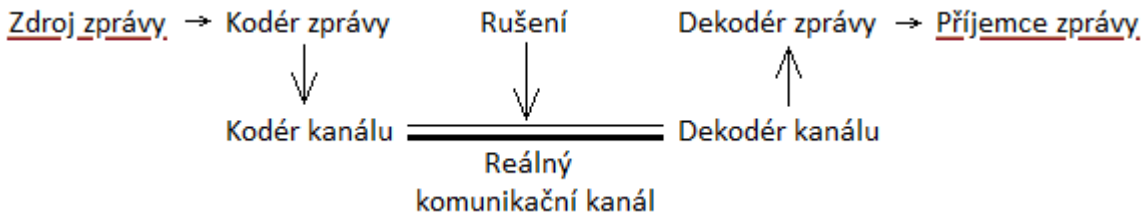
Hammingova vzdálenost je dále využita při dekódování konvolučních kódů pro určování nejpravděpodobnějšího odhadu zdrojové informace.

1.3 Přenos informací

Před přenosem zdrojové informace reálným komunikačním kanálem musí být tato informace zakódována kodérem do posloupnosti bitů, která je přenášena přes komunikační kanál. Protože se tento přenos děje v reálném čase přes reálný komunikační kanál, působí na zakódovanou posloupnost okolní vlivy, popřípadě vlastnosti materiálu komunikačního kanálu a dochází ke vzniku chyb v zakódované posloupnosti. Zvolením vhodné metody pro dekódování posloupnosti můžeme chyby eliminovat.

Protože je komunikační kanál bezpaměťový, závisí bezchybný přenos informace pouze na aktuálně přenášeném znaku, nikoli na několika předchozích znacích. [2] a [4]

Celý pochod informace od zdroje až k příjemci je zobrazen na obrázku (Obr. 1-3).



Obr. 1-3. Komunikační kanál pro přenos informací

1.4 Binární operace modulo 2

Při práci s binárními lineárními kódy se využívá matematických operací, a to *sečítání modulo 2*, uveden v tabulce (Tab. 1.), jejíž symbol je $\oplus$ a *násobení modulo 2*, uveden v tabulce (Tab. 2.), jejíž symbol je $\otimes$. [2] a [3]

Tab. 1-1. Sčítání modulo 2

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

Tab. 1-2. Násobení modulo 2

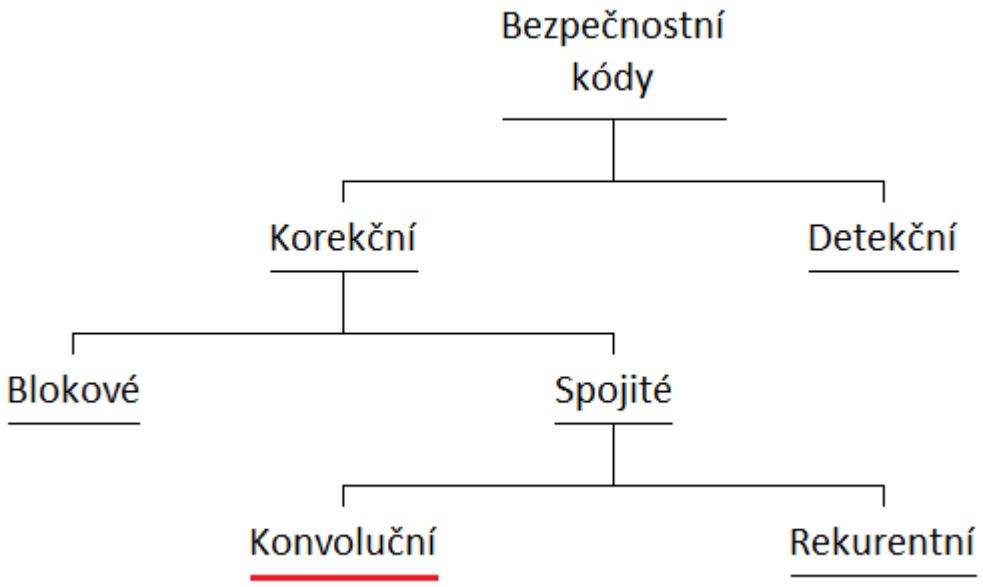
A	B	A ⊗ B
0	0	0
0	1	0
1	0	0
1	1	1

1.5 Rozdělení kódů

Účelem této práce je seznámení s konvolučním kódováním, proto je ukázáno jejich zařazení.

Bezpečnostní kódy se dělí na dvě hlavní skupiny, jsou to kódy *korekční* a *detekční*. *Detekční* kódy umí chybu v kódu pouze nalézt, odtud tedy detekční, kdežto kódy *korekční* umí chybu jak nalézt, tak i opravit. Počet detekcí a oprav chyb v kódu se liší u každého druhu kódu. Korekční kódy se dále dělí na kódy *blokové* a *spojité*. Do kódů *blokových* patří například kódy cyklické, Hammingovy, Golayův kód, BCH kód a jiné. Do kódů *spojitých* potom patří například kódy rekurentní, řetězové a jiné. [1], [3] a [4]

Do kódů spojitých také patří konvoluční kódy, kterými se bude práce nadále zabývat.



Obr. 1-4. Schéma zařazení konvolučních kódů

2 KONVOLUČNÍ KÓDY

2.1 Základní pojmy

2.1.1 Konvoluce

Konvoluce je matematická operace, jako je sčítání, odčítání, násobení či dělení. Konvoluce je sloučení dvou jednorozměrných funkcí *f(x)* a *g(x)* do třetí výsledné, značí se operandem „*“ a je dána vztahem:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\alpha) g(x - \alpha) \mathrm{d}\alpha \tag{2.1}$$

Funkce *f(x)* reprezentuje zdrojovou posloupnost, někdy označovanou také jako informační posloupnost, která vstupuje do kodéru. Funkce *g(x)* je nazývána *jádrem konvoluce,* reprezentuje generující polynom, dle kterého se vytváří kodér. Výsledkem konvoluce je tzv. kódová posloupnost.

Konvoluce se také využívá při zpracování obrazu, více v literatuře [8].

2.1.2 Konvoluční kód, konvoluční kodér, konvoluční kódování

Konvoluční kód a konvoluční kodér spolu úzce souvisí.

Konvoluční kód – soubor kódových posloupností, které představují všechny možnosti zdrojových posloupností *f(x).*

Konvoluční kodér – stroj nebo součástka, dána generujícím mnohočlenem *g(x).* Do kodéru vstupuje zdrojová posloupnost a ta je kodérem zakódována do posloupnosti kódové.

Konvoluční kódování – činnost, kterou provádí kodér, když převádí vstupní informační posloupnost na výstupní kódovou posloupnost.

O konvolučních kódech se zle podrobně dočíst v literaturách [2] a [5].

2.2 Konvoluční kodér

Konvoluční kodéry řadíme mezi kodéry s pamětí. Výstupní kódová posloupnost není dána jen aktuální vstupní informační posloupností, ale závisí také na *m* předešlých vstupních informačních posloupností.

Konvoluční kód je obecně zapisován jako konvoluční *(n, k)*-kód. Tento zápis značí, že *k*-bitová informace bude kodérem zakódována do *n*-bitového slova. Z pravidla je dáno, že $n \geq k$.

2.2.1 Vlastnosti konvolučního *(n, k)*-kódu

Pro konvoluční *(n, k)*-kód je dán předpis *K*, který každému polynomu *u(x)* přiřadí polynom *K[u(x)]*.

Základní vlastnosti kódu *K[u(x)]*:

Linearita:

$$K[u(x) + u'(x)] = K[u(x)] + K[u'(x)] \tag{2.2}$$

a

$$K[tu(x)] = tK[u(x)] \tag{2.3}$$

Časová invariantnost:

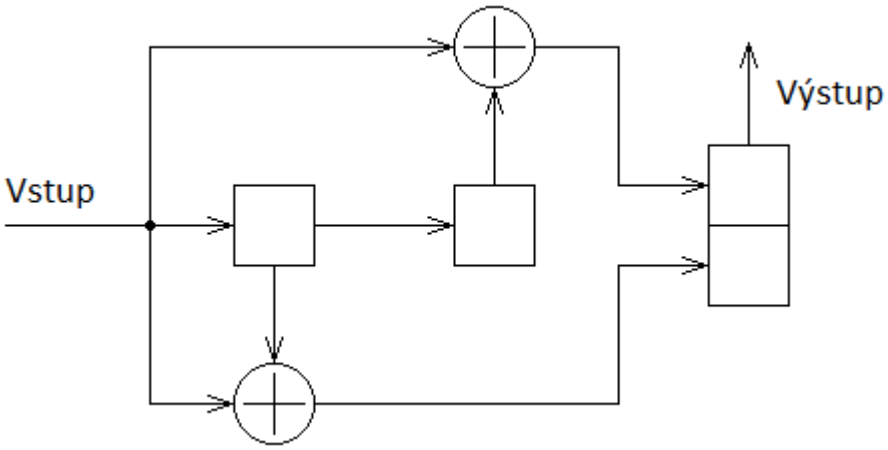
Kód *K* je časově invariantní, když časové zpoždění o *k* bitů u zdrojového slova vyvolá časové zpoždění o *n* bitů u kódového slova.

$$K[x^k u(x)] = x^n K[u(x)] \tag{2.4}$$

Počet *k*-tic zdrojových symbolů ovlivňujících jednu *n*-tici kódových symbolů udává významný parametr konvolučních kódů, tzv. *omezující délka* (constraint length). [2]

Příklad 2.1:

Linearita a časová invariantnost je ukázána na příkladu. Mějme kodér konvolučního *(2, 1)*-kódu daný generujícím polynomem $g(x) = 1 + x + x^3 + x^4$.



Obr. 2-1. Kodér konvolučního (2, 1)-kódu s g(x) = 1 + x + x^3 + x^4

Při průchodu signálu kodérem je využito operace sčítání modulo 2 značené symbolem $\oplus$. V počátečním stavu mají posuvné registry nulový obsah. Je-li na vstup kodéru poslaná jedna 1, je na výstupu získána odezva ve stavu 11. V dalším kroku je na vstup poslána jedna 0, první posuvný registr si pamatuje předešlý stav vstupu, tzn., že je tento registr nastavený na hodnotu 1. Druhý posuvný registr si pamatuje předešlý stav prvního registru, tzn., že jeho obsah je opět nulový. Po průchodu nuly takto nastaveným kodérem je na výstupu získána hodnota 01. Ve třetím kroku opět pošleme na vstup kodéru 0. První registr je tentokrát nulový a druhý registr obsahuje předešlou hodnotu prvního registru, tzn., že je nastavený na hodnotu 1. Po průchodu nuly registrem je na výstupu získána hodnota 10. Po tomto kroku mají oba posuvné registry nulový obsah a kódování je ukončeno. Použitím mnohočlenů lze operaci zapsat následovně:

$$K[1] = 110110 = 1 + x + x^3 + x^4 \tag{2.5}$$

Použití druhé vlastnosti kódu, časové invariantnosti, je ukázáno na následujícím:

$$K[01] = 00110110 \tag{2.6}$$

$$K[001] = 0000110110 \tag{2.7}$$

Jak je vidět, časové zpoždění o 1 bit u zdrojového slova vyvolá časové zpoždění o 2 bity u kódového slova u vztahu (2.6).

Použití první vlastnosti kódu, linearity, je ukázáno na následujícím:

$$K[101] = K[1] + K[001] \tag{2.8}$$

Ověření:

Je-li na vstup kodéru konvolučního *(2, 1)*-kódu s generujícím polynomem $g(x) = 1 + x + x^3 + x^4$ přivedena informační posloupnost $f(x) = 1 + x^2$ = 101, je následně na výstupu kódovou získána posloupnost ve tvaru 1101010110.

Dle vlastností kódu *K[u(x)]*, linearity a časové invariantnosti, platí:

K[1] = 1 1 0 1 1 0

\+ K[001] = 0 0 0 0 1 1 0 1 1 0

K[101] = 1 1 0 1 0 1 0 1 1 0

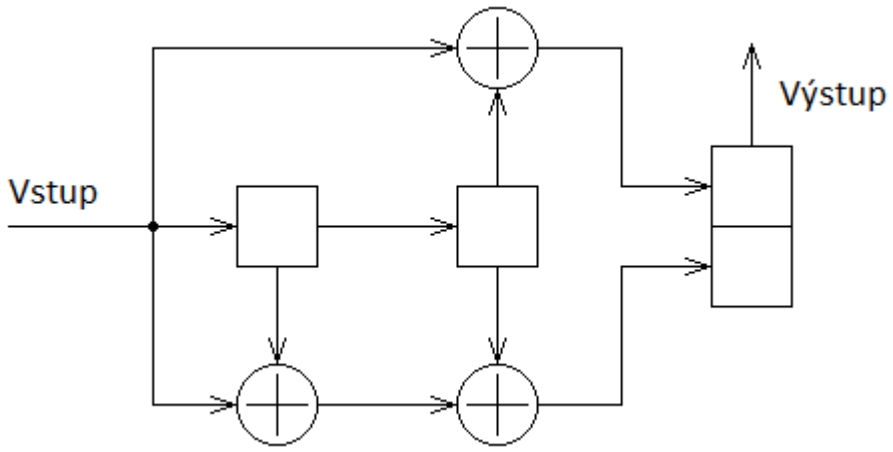
Jak je vidět, součet K[1] + K[001] je shodný s kódovou posloupností na výstupu kodéru při informační posloupnosti 101 na vstupu kodéru.

2.2.2 Kodér konvolučního *(n, k)*-kódu

Konvoluční kodér bývá složen z klopných obvodů posuvného registru, obvodů pro sčítání modulo 2 a *n*-bitové výstupní vyrovnávací paměti. Jelikož jsou použity pouze lineární prvky, je mezi zdrojovou posloupností a kódovou posloupností lineární vztah. [2]

Příklad 2.2:

Pro příklad je uveden kodér konvolučního *(2, 1)*-kódu s generujícím polynomem $g(x) = 1 + x + x^3 + x^4 + x^5$.



Obr. 2-2. Kodér konvolučního (2, 1)-kódu s $g(x) = 1 + x + x^3 + x^4 + x^5$

Tento kodér se skládá ze dvou klopných obvodů posuvného registru, ze tří obvodů pro sčítání modulo 2 a z dvoubitové výstupní vyrovnávací paměti.

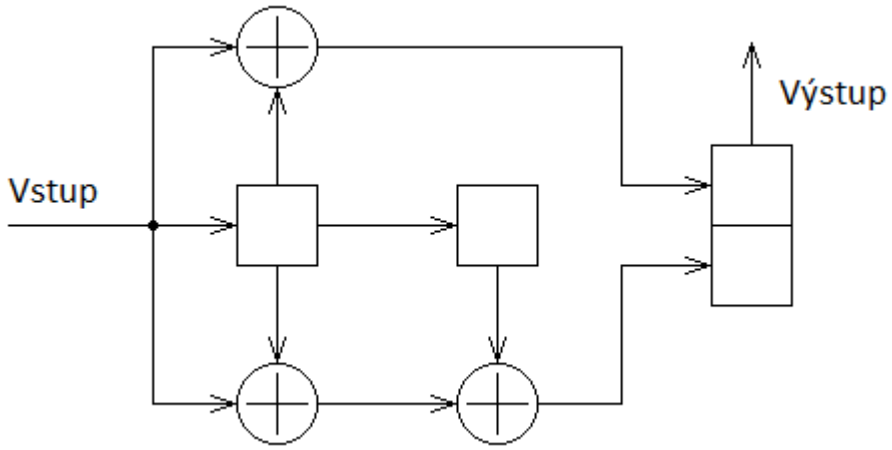
2.2.3 Generující mnohočlen, generující matice

Jak už bylo řečeno, konvoluční kodér na vstupu přijímá informační posloupnost v čase *i* a na výstupu je získána odezva, kódová posloupnost, také v čase *i*. Ovšem, také bylo řečeno, že konvoluční kódy jsou kódy s pamětí, tzn., že výsledná kódová posloupnost není odezvou jen na informační posloupnost v čase *i*, ale také na předešlých informacích v čase *j*, kde $j \neq i$.

Vztah mezi informační posloupností a kódovou posloupností je dán generujícím mnohočlenem *g(x)*. [5]

Generující mnohočlen je získán tak, že je na vstup kodéru přivedena jedna 1. V dalších krocích jsou na vstup přiváděny nuly, dokud nemá kodér všechny posuvné registry opět nulové.

Příklad 2.3:



Obr. 2-3. Kodér konvolučního (2, 1)-kódu s $g(x) = 1 + x + x^2 + x^3 + x^5$

Na vstup kodéru je přivedena jedna 1. První i druhý posuvný registr jsou nulové, tudíž na výstupu je získáno slovo 11. V dalším kroku je na vstup přivedena 0, v prvním registru je uložena hodnota 1 a ve druhém registru je stále hodnota 0. Na výstupu je opět získáno slovo 11. Ve třetím kroku je opět na vstup kodéru přivedena 0, první registr je

nulový a ve druhém je uložena hodnota 1. Na výstupu kodéru je získáno slovo 01. Po tomto kroku se oba posuvné registry nachází opět v nulovém stavu a kódování tímto končí. Výsledná kódová posloupnost odpovídá generujícímu mnohočlenu. Zápis pomocí mnohočlenů vypadá následovně:

$$K[1] = 111101 = 1 + x + x^2 + x^3 + x^5 \tag{2.9}$$

$$\Rightarrow \qquad g(x) = 1 + x + x^2 + x^3 + x^5 \tag{2.10}$$

V příkladě bylo ukázáno, jak získat generující mnohočlen ze stavby kodéru.

S generujícím polynomem *g(x)* úzce souvisí generující matice *G*. Řádek generující matice tvoří generující polynom. Další řádek matice je vždy posunut o *n* pozic doprava. Pokud je kodér zadán více generujícími polynomy ($g_0(x)$, $g_1(x)$ ... $g_n(x)$), tvoří každý generující polynom jeden řádek matice. Řádky se posouvají o *n* pozic doprava až po zadání všech generujících polynomů. Generující matice není ukončena, protože není předem známo, jaká informační posloupnost bude na vstup přivedena. Počet řádků matice je dán počtem informačních bitů.

Více o generujícím polynomu a generující matici se lze dočíst v literatuře [2] a [3].

Obecná generující matice:

$$G = \begin{pmatrix} g_0(x) & & & \\ g_1(x) & & & \\ \vdots & & & \\ g_n(x) & & & \\ & g_0(x) & & \\ & g_1(x) & & \\ & \vdots & & \\ & g_0(x) & & \\ & & g_0(x) & \\ & & g_1(x) & \\ & & \vdots & \\ & & g_0(x) & \end{pmatrix} \tag{2.11}$$

Příklad 2.4:

Pro kodér konvolučního *(2, 1)*-kódu s generujícím polynomem $g(x) = 1 + x + x^3 + x^5 = 110101$ vypadá generující matice následovně:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & \dots & \dots \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & & \\ & & & & & \vdots & & & & & \ddots & \\ & & & & & \vdots & & & & & & \ddots \end{pmatrix} \tag{2.12}$$

Protože tento kodér generuje na výstupu v každém kroku dvoubitové slovo, tak i každý následující řádek generující matice je posunut o dvě pozice doprava oproti předchozímu řádku.

Je-li na vstup kodéru přivedena informační posloupnost, například $f(x) = 11001$, má pak generující matice pět řádků, protože je na vstup přivedena pětibitová informace. Tato generující matice bude mít potom tvar:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{2.13}$$

Příklad 2.5:

Pro kodér konvolučního *(2, 2)*-kódu s generujícími polynomy $g_0(x) = 1 + x + x^3 + x^5 = 110101$ a $g_1(x) = 1 + x2 + x4 + x5 = 101011$ vypadá generující matice následovně:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & & \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & & \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & \dots & \dots \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & & \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & & \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & & \\ & & & & & \vdots & & & & & \ddots & \\ & & & & & \vdots & & & & & & \ddots \end{pmatrix} \tag{2.14}$$

2.2.4 Sestavení kodéru konvolučního *(n, k)*-kódu

V předešlé kapitole bylo ukázáno, jak získat generující mnohočlen ze stavby kodéru. Stejně je možné ze zadaného generujícího mnohočlenu sestavit kodér. Kodér se sestaví tak, že když je na vstup kodéru přivedena jedna 1, na výstupu musí být získán generující mnohočlen. Následující příklad ukazuje postup při vytváření kodéru z generujícího mnohočlenu.

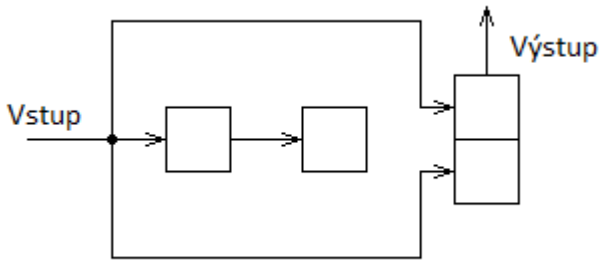
Příklad 2.6:

Úkolem bude sestavit kodér konvolučního *(2, 1)*-kódu, který je využit v příkladu (Příklad 2.2) s generujícím mnohočlenem $g(x) = 1 + x + x^3 + x^4 + x^5 = 110111$, pro ověření správnosti sestavení.

V prvním kroku je generující mnohočlen rozdělendo *n*-tic bitů, v tomto případě do dvojic bitů, následovně:

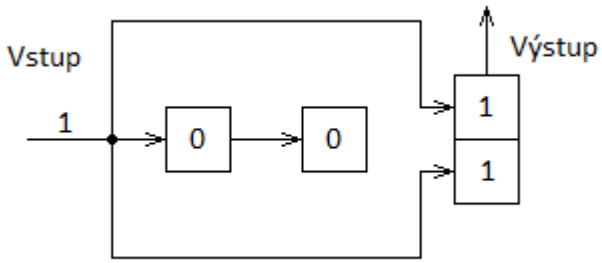
$$g(x) = 11 \mid 01 \mid 11 \tag{2.15}$$

V druhém kroku je sestavena základní kostra kodéru s příslušným počtem vstupů, výstupů a posuvných registrů. V tomto případě kodér obsahuje jeden vstup, dva výstupy a dva posuvné registry. Počet posuvných registrů vždy volíme o jeden méně, než je počet *n*-tic. V tomto případě jsou 3 dvojice, proto posuvné registry jsou 2.



Obr. 2-4. Základní kostra kodéru

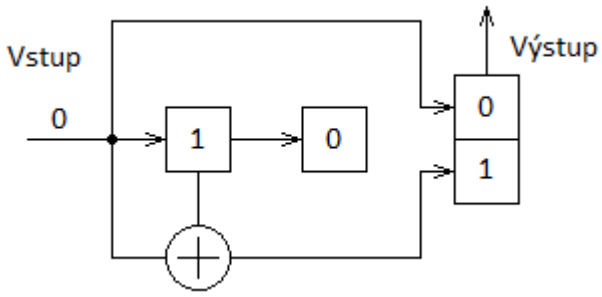
Na vstup je přivedena jedna 1. Na výstupu musí být získána odezva ve stejném stavu, jaká je první dvojice bitů z generujícího mnohočlenu, tudíž 11. Hodnoty obou registrů jsou vždy znázorněny na obrázku.



Obr. 2-5. První krok

Jak lze vidět, na výstupu tuto hodnotu dostáváme a prozatím se kodér nemění.

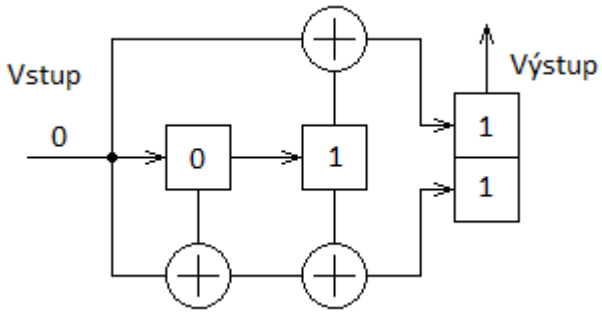
Na vstup kodéru je přivedena 0. Na výstupu musí být získán stejný stav, jaká je druhá dvojice bitů z generujícího mnohočlenu, tj. 01.



Obr. 2-6. Druhý krok

„Horní“ větev obvodu zůstává stejná a na výstupu je hodnota 0. „Dolní“ větev obvodu ovšem musí být spojena s prvním posuvným registrem za pomocí členu pro sčítání modulo 2 a využito pravidel pro sčítání modulo 2. Dle tabulky (Tab. 1-1) platí, že součet čísel 0 a 1 je ve výsledku 1 a na výstupu je hodnota 1. Výstupní slovo pro tento krok je potom 01.

Následně na vstup opět přivedeme 0. Na výstupu musí být získán stejný stav, jaká je třetí dvojice bitů z generujícího mnohočlenu, tj. 11.



Obr. 2-7. Třetí krok

„Horní“ větev obvodu je spojena s druhým posuvným registrem za pomocí členu pro sčítání modulo 2 a na výstupu je získána hodnota 1. „Dolní“ větev obvodu je také spojena s druhým posuvným registrem za pomocí členu pro sčítání modulo 2 a na výstupu je také získána hodnota 1. Výstupní slovo pro tento krok je potom 11.

Tímto stavba kodéru končí. Jak lze vidět, získaný kodér opravdu souhlasí s kodérem z příkladu (Příklad 2.2).

2.3 Kódování konvolučního kódu

Převodu informační posloupnosti na kódovou posloupnost se říká kódování. Kódování je prováděno kodérem, který je dán generujícím polynomem a generující polynom je základem generující matice. Z toho zle vidět, že vše spolu úzce souvisí.

Kódování informační posloupnosti do posloupnosti kódové je dáno vztahem:

$$v(x) = f(x) * G \tag{2.16}$$

Kde: *v(x)* – výstupní kódová posloupnost

f(x) – vstupní informační posloupnost

G – generující matice

Při násobení informační posloupnosti maticí je využito pravidel pro násobení modulo 2, které se značí symbolem $\otimes$ a pravidel pro sčítání modulo 2, které se značí symbolem $\oplus$.

Příklad 2.7:

Mějme kodér konvolučního *(2, 1)*-kódu s generujícím polynomem $g(x) = 1 + x + x^3 + x^5$ = 110101. Na vstup je přivedena informační posloupnost $f(x) = 1 + x^2 + x^3$ = 1011.

Generující matice takového kodéru bude mít tvar:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{2.17}$$

Výstupní kódová posloupnost se potom vypočítá dle vztahu (2.14):

$$v(x) = (1 \quad 0 \quad 1 \quad 1) * \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{2.18}$$

$$v(x) = 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \tag{2.19}$$

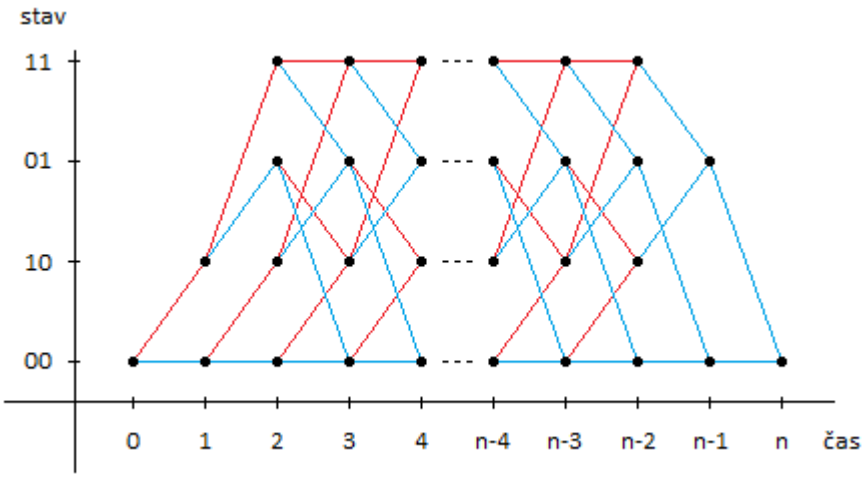
Oproti maticovému vyjádření se častěji využívá grafického popisu kódování. Ke grafickému popisu slouží tzv. *mřížkový diagram* (trellis diagram). Je to ekvivalent k úplnému kódovému stromu, který má sloučené shodné stavy. Začátek a konec mřížkového diagramu je znázorněn na obrázku (Obr. 2-8). Přechody mezi stavy jsou nazývány *cesta* (path). Na počátku kódování má kodér všechny posuvné registry v nulovém stavu, proto i mřížkový diagram začíná v nule. Po „průchodu“ zdrojové posloupnosti je na kodér přivedeno tolik nul, aby se jeho posuvné registry opět nacházely v nulovém stavu. Tyto nuly se nazývají *konec zprávy* (tail of the message) a jejich počet je:

$$k(K-1), \tag{2.20}$$

kde: k – počet vstupů do kodéru

K – omezující délka konvolučního kódu

Možná podoba mřížkového diagramu pro konvoluční *(2, 1)*-kodér s dvojicí pamětí je na následujícím obrázku.

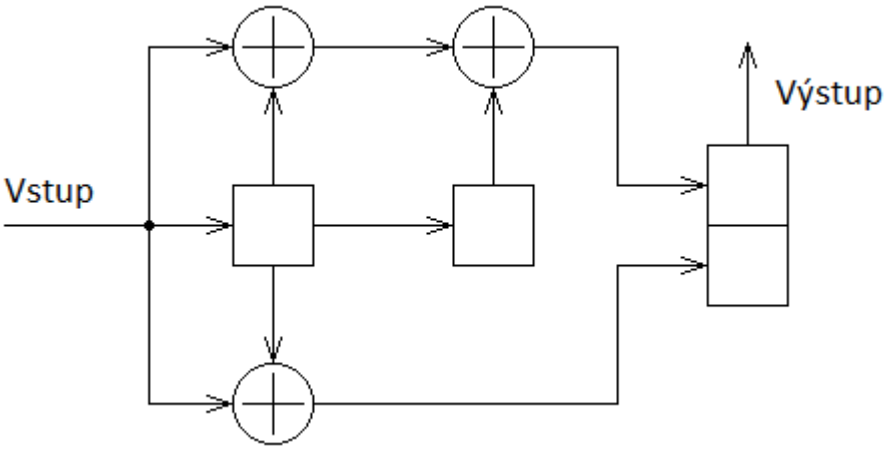


Obr. 2-8. Mřížkový diagram

Modře jsou označeny cesty při příchodu nuly na vstup kodéru, červeně cesty při příchodu jedničky na vstup kodéru.

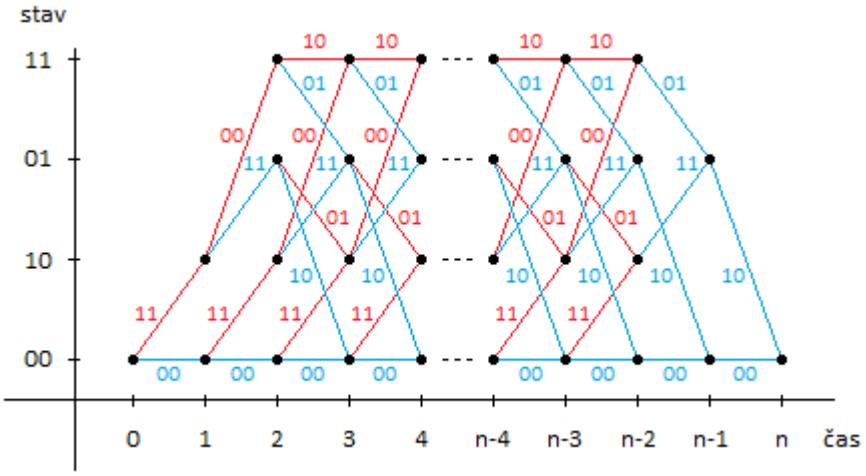
Příklad 2.8:

Mějme kodér konvolučního *(2, 1)*-kódu s $g(x) = 1 + x + x^2 + x^3 + x^4 = 111011$. Na vstup je přivedena zdrojová posloupnost $f(x) = 1 + x^2 + x^3 = 1011$.



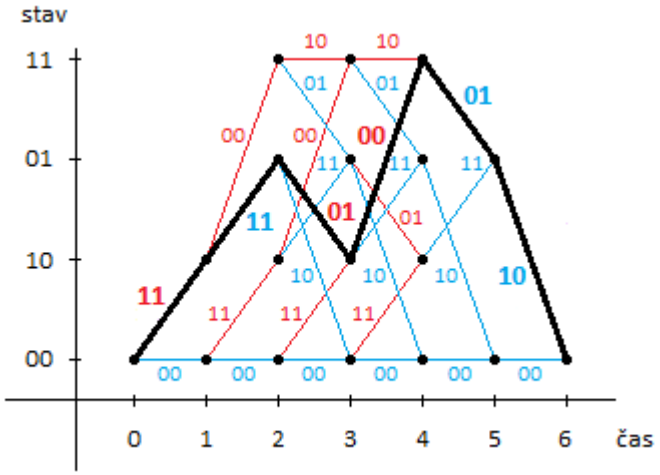
Obr. 2-9. Kodér konvolučního (2, 1)-kódu s $g(x) = 1 + x + x^2 + x^3 + x^4$

Pro zakódování zdrojové posloupnosti je využito mřížkového diagramu z obrázku (Obr. 2-8), do něhož jsou vyznačeny i stavy výstupů po příchodu nuly nebo jedničky na vstup kodéru.



Obr. 2-10. Mřížkový diagram konvolučního (2, 1)-kódu s generujícím mnohočlenem $g(x) = 1 + x + x^2 + x^3 + x^4$

Výsledná cesta tohoto kodéru je na obrázku (Obr. 2-11). Ta je získána zvýrazněním cesty mezi dvěma stavy, která odpovídá aktuální hodnotě na vstupu kodéru. Na konec je na kodér přiveden potřebný počet nul podle vztahu (2.18). Zdrojová posloupnost je 1011, kódová posloupnost je potom 111101000110.



Obr. 2-11. Výsledná cesta konvolučního (2, 1)-kódu zobrazující kódovou posloupnost

2.4 Dekódování konvolučního kódu

Při průchodu kódové posloupnosti reálným komunikačním kanálem dochází ke zkreslení signálu a tím i vzniku chyb v kódové posloupnosti.

Dekódování konvolučního kódu už není tak „jednoduché“ jako je kódování. Výsledná cesta převážně nemůže být určována ihned, ale až na základě několika následujících kroků. Proto se výsledná cesta, a tím i dekódovaná informační posloupnost, určují až na konci dekódování a to na základě *nejpravděpodobnějšího odhadu*.

Nejpoužívanější algoritmus pro dekódování, tzn. nalezení nejpravděpodobnějšího odhadu informační posloupnosti, je *Viterbiho algoritmus*, kterému se věnují i literatury [2] a [5].

2.4.1 Viterbiho algoritmus

Viterbiho algoritmus nachází nejpravděpodobnější informační posloupnost procházením prostoru kódových posloupností. Počet kódových posloupností roste exponenciálně s omezující délkou K konvolučního kódu, proto se Viterbiho algoritmus využívá maximálně do délky $K = 10$. S omezující délkou K konvolučního kódu je spjat i počet numerických operací užitých u Viterbiho algoritmu. Jejich počet, pro dekódování zdrojové posloupnosti o délce L, je:

$$L2^{k(K-1)} \tag{2.21}$$

kde: k – počet vstupů do kodéru

K – omezující délka konvolučního kódu

Ze vztahu (2.19) lze vidět, že jejich počet také s rostoucí délkou K konvolučního kódu roste exponenciálně.

Viterbiho algoritmus pracuje s mřížkovým diagramem (Obr. 2-8) a využívá Hammingovy minimální vzdálenosti. Přijatá kódová posloupnost je rozdělena do n-tic. V každém kroku v čase i se pak porovnávají všechny možné stavy výstupů s i-tou n-ticí, tzn., v prvním kroku v čase jedna se výstupy porovnávají s první n-ticí, ve druhém kroku s druhou n-ticí atd. Výsledkem porovnání je Hammingova vzdálenost. V každém kroku do jednoho stavu vstupuje 2^k cest a vychází také 2^k cest, tzn., že se na každém stavu „objeví“ 2^k Hammingových vzdáleností. Danému stavu se zapíše nejmenší z 2^k vzdáleností. Pokud

do jednoho stavu vstoupí více cest se stejnou Hammingovou vzdáleností, potom je možné vybrat náhodně jednu z nich. Poté se vybírá nejmenší Hammingova vzdálenost ze všech stavů, která určuje cestu a říká, že daná *n*-tice je nejblíže k přijaté kódové posloupnosti. V případě, že je více stejných vzdáleností, je výsledná cesta určena až na základě několika dalších kroků. Výsledná cesta určuje nejpravděpodobnější odhad původní informační posloupnosti.

Postup dekódování je ukázán na dvou následujících příkladech, kdy je v prvním případě obdržena kódová posloupnost bez chyby a ve druhém případě s chybou

Příklad 2.9:

V tomto příkladě je využito příkladu (Příklad 2.8). Na vstup kodéru konvolučního *(2, 1)*-kódu s generujícím polynomem $g(x) = 1 + x + x^2 + x^3 + x^4$, jenž je na obrázku (Obr. 2-9), je přivedena zdrojová posloupnost $f(x) = 1 + x^2 + x^3 = 1011$. Ta byla zakódována do posloupnosti 111101000110. Při přenosu nedošlo k žádným chybám.

V prvním kroku je přijatá kódová posloupnost rozdělena do *n*-tic bitů, v tomto případě do dvojic bitů, následovně:

11 | 11 | 01 | 00 | 01 | 10

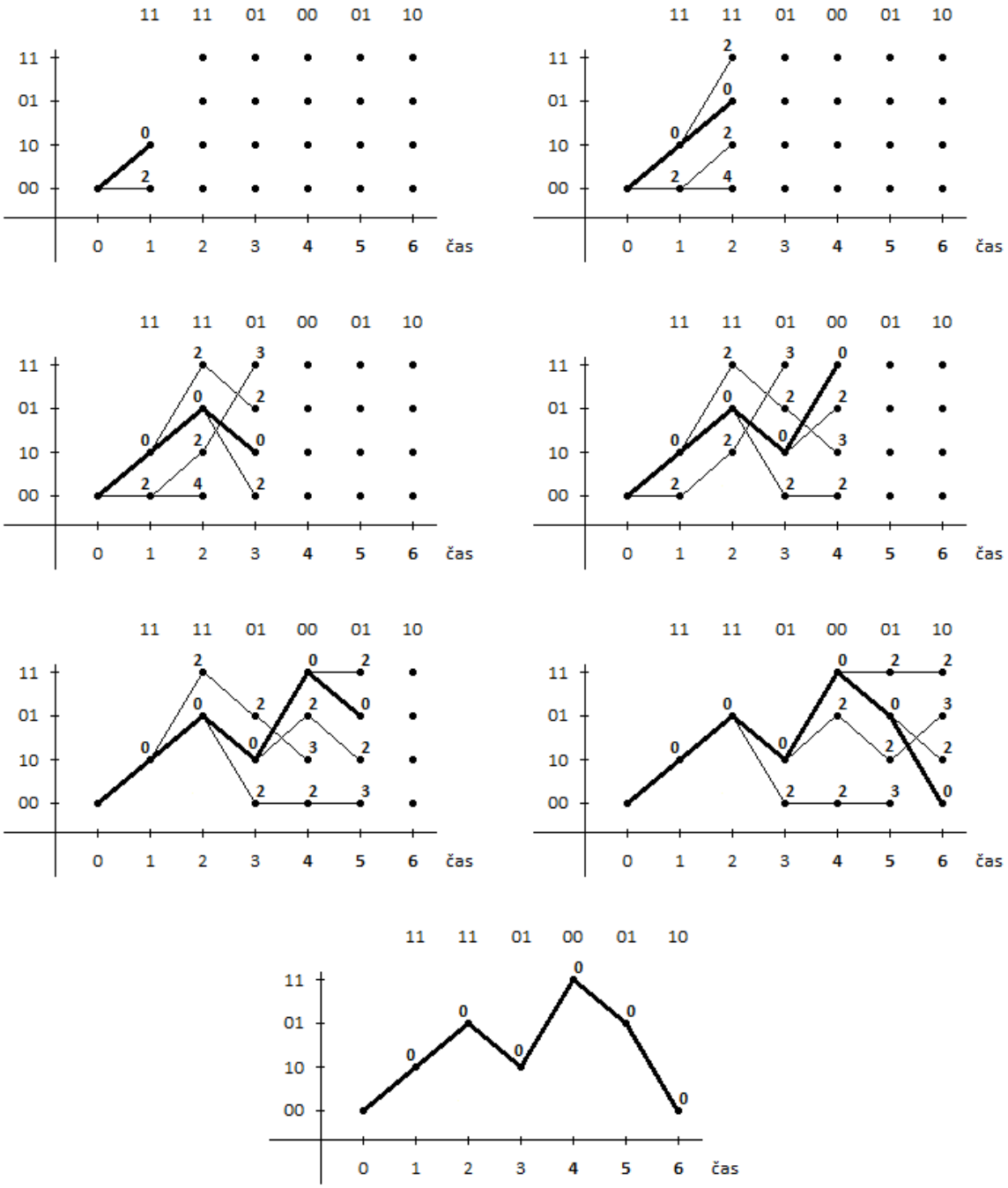
Pro dekódování je využito mřížkového diagramu (Obr. 2-8). Nad každý krok (čas) je napsána přijatá dvojice kódového slova. Aby byl diagram přehlednější, nebudou do něj tentokrát vepsány hodnoty výstupů při příchodu jedničky nebo nuly na vstup kodéru, ale bude využita tabulka (Tab. 2-1), kde jsou vypsané všechny možné kombinace výstupů.

Tab. 2-1. Tabulka stavů registrů a výstupu

Původní tav registrů	Vstupní bit	Nový stav registrů	Výstup
00	0	00	00
00	1	10	11
10	0	01	11
10	1	11	00
01	0	00	10
01	1	10	01
11	0	01	01
11	1	11	10

Tabulka (Tab. 2-1) ukazuje, jak bude vypadat výstupní slovo s danými stavy posuvných registrů a při příchodu jedničky nebo nuly na vstup kodéru a také jaké hodnoty budou posuvné registry nabývat po tomto kroku. Tabulka je vytvořena pro zadaný kodér konvolučního *(2, 1)*-kódu, jenž je na obrázku (Obr. 2-9).

Následující posloupnost diagramů zobrazuje průběh dekódování.



Obr. 2-12. Průběh odhadu nejpravděpodobnější podoby zdrojové posloupnosti bez chybového přenosu

Jelikož byl přenos informací bezchybný, bylo dekódování snadné. Výsledná cesta, tzn. nejpravděpodobnější odhad zdrojové posloupnosti, byla dána nulovými Hammingovými vzdálenostmi. Jak lze vidět, je výsledná cesta stejná jako na obrázku (Obr. 2-11), což svědčí o správnosti dekódování.

Podrobný průběh při dekódování je uveden v tabulce (Tab. 2-2).

Tab. 2-2. Dekódovací tabulka bez chybového přenosu

Stav registrů	Přijatá posloupnost	11	11	01	00	01	10
11	x	x	2 (1)	~~4 (1)~~	~~4 (1)~~	2 (1)	2 (1)
				3 (1)	**0 (1)**	~~3 (1)~~	~~3 (1)~~
01	x	x	**0 (0)**	2 (0)	~~4 (0)~~	**0 (0)**	~~4 (0)~~
				~~3 (0)~~	2 (0)	~~4 (0)~~	3 (0)
10	x	**0 (1)**	2 (1)	**0 (1)**	3 (1)	2 (1)	2 (1)
				~~5 (1)~~	~~4 (1)~~	~~3 (1)~~	~~4 (1)~~
00	x	2 (0)	4 (0)	2 (0)	~~3 (0)~~	~~4 (0)~~	**0 (0)**
				~~5 (0)~~	2 (0)	3 (0)	~~4 (0)~~
Čas	0	1	2	3	4	5	6

Hodnoty před závorkami jsou Hammingovy vzdálenosti, hodnoty v závorkách označují bit přivedený na vstup kodéru a zvýrazněná čísla označují průběh výsledné cesty. Sepsáním čísel v závorce u zvýrazněných čísel je získána zdrojová posloupnost, tedy:

$$101100 => f(x) = 1011 = 1 + x^2 + x^3 \tag{2.22}$$

Zdrojová posloupnost byla dekódována správně.

Příklad 2.10:

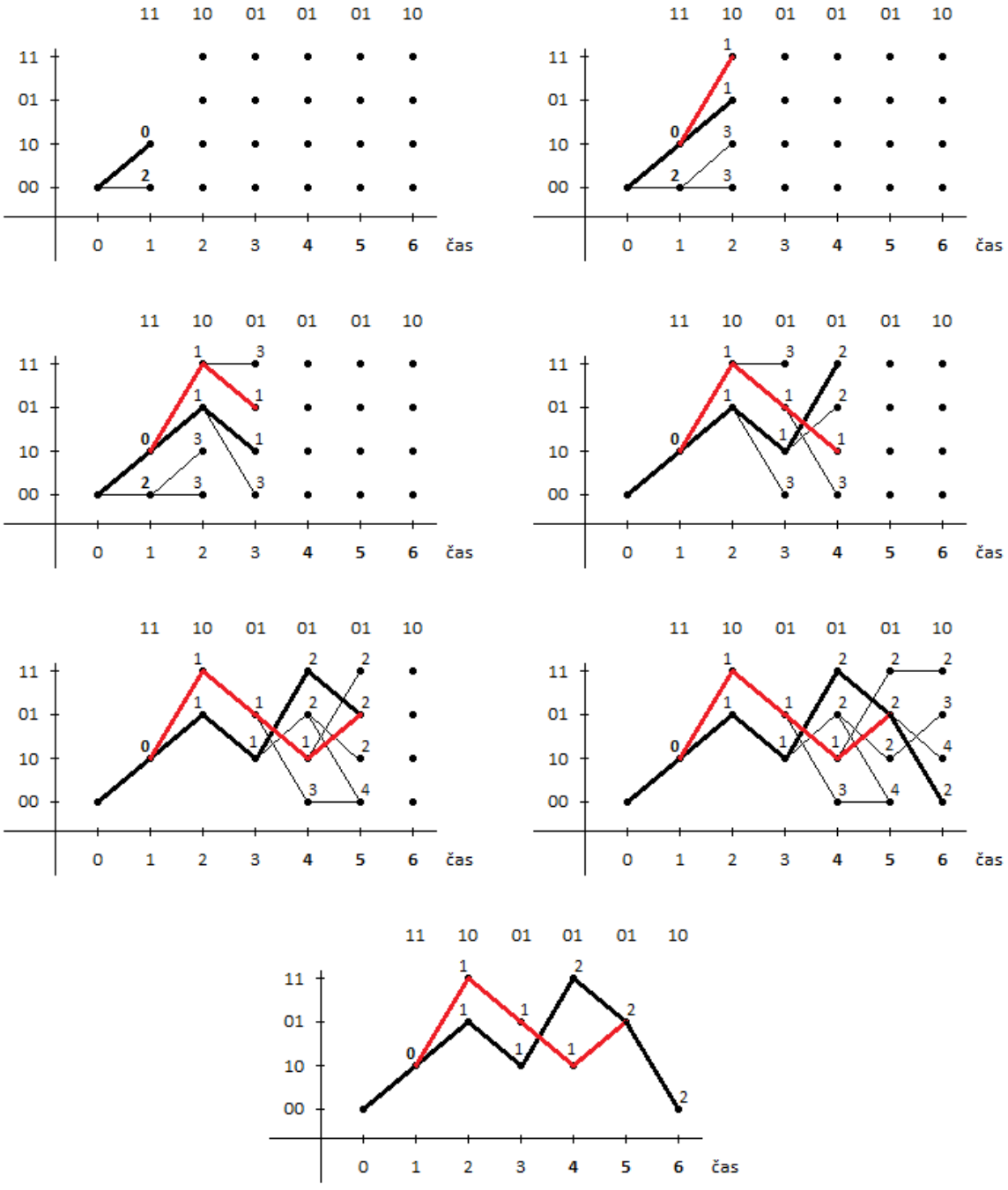
Nyní bude ukázán příklad, kdy při přenosu informace došlo ke dvěma chybám. Opět bude využito příkladu (Příklad 2.8). Zdrojová posloupnost 1011 byla zakódována do posloupnosti 111101000110, ovšem přijata byla posloupnost 111001010110.

V prvním kroku je opět přijatá kódová posloupnost rozdělena do dvojic bitů, následovně:

11 | 10 | 01 | 01 | 01 | 10

Pro dekódování je využito mřížkového diagramu (Obr. 2-8). Nad každý krok (čas) je napsána přijatá dvojice kódového slova. Opět je využita tabulka (Tab. 2-1), kde jsou vypsané všechny možné kombinace výstupů.

Následující posloupnost diagramů zobrazuje průběh dekódování.



Obr. 2-13. Průběh odhadu nejpravděpodobnější podoby zdrojové posloupnosti chybového přenosu

Jelikož byl přenos informací chybový, bylo dekódování obtížnější. Výsledná cesta, tzn. nejpravděpodobnější odhad zdrojové posloupnosti, nemohla být určována okamžitě, ale až na základě několika dalších kroků. V kroku 2 byly získány dvě stejné minimální Hammingovy vzdálenosti a tedy dvě možnosti postupu, první varianta je značena dále černou barvou, druhá alternativní varianta je značená červenou barvou, proto nebylo možné hned určit, která cesta je správná. V kroku 5 se obě cesty opět spojily se stejnými Hammingovými vzdálenostmi a po dekódování celého slova měly obě varianty cest stejnou Hammingovu vzdálenost, proto není možné jednoznačně určit, která je správná a na které pozici vznikly při přenosu informace chyby. První chyba v obou případech vznikla v kroku 2. Druhá chyba v případě první varianty (černá cesta) vznikla již v kroku 4, kdežto v případě druhé varianty (červená cesta) vznikla až v kroku 5.

Podrobný průběh při dekódování je uveden v tabulce (Tab. 2-3).

Tab. 2-3. Dekódovací tabulka chybového přenosu

Stav registrů	Přijatá posloupnost	11	10	01	01	01	10
11	x	x	**1 (1)**	3 (1) ~~4 (1)~~	~~5 (1)~~ **2 (1)**	~~4 (1)~~ 2 (1)	2 (1) ~~3 (1)~~
01	x	x	**1 (0)**	**1 (0)** ~~4 (0)~~	~~3 (0)~~ 2 (0)	**2 (0)** **2 (0)**	~~4 (0)~~ 3 (0)
10	x	**0 (1)**	3 (1)	**1 (1)** ~~4 (1)~~	**1 (1)** ~~4 (1)~~	2 (1) ~~4 (1)~~	4 (1) ~~5 (1)~~
00	x	2 (0)	3 (0)	3 (0) ~~4 (0)~~	3 (0) ~~4 (0)~~	4 (0) 4 (0)	**2 (0)** ~~5 (0)~~
Čas	0	1	2	3	4	5	6

Hodnoty před závorkami jsou Hammingovy vzdálenosti, hodnoty v závorkách označují bit přivedený na vstup kodéru a zvýrazněná čísla označují průběh výsledné cesty. Černá čísla označují první variantu odhadu, červená čísla označují druhou variantu odhadu. Sepsáním čísel v závorce u zvýrazněných čísel je získána zdrojová posloupnost, tedy:

$$1.\, varianta \rightarrow 101100 => f(x) = 1011 = 1 + x^2 + x^3 \qquad (2.23)$$

$$2.\, varianta \rightarrow 110100 => f(x) = 1101 = 1 + x + x^3 \qquad (2.24)$$

Jak je vidět, byly získány dva možné výsledky, dvě možné zdrojové posloupnosti, což dokazuje, že dekódováním konvolučního kódu není získán stoprocentní výsledek, ale

jen nejpravděpodobnější odhady původní zdrojové posloupnosti. Výše bylo uvedeno, že pokud do jednoho stavu vstoupí více cest se stejnou Hammingovou délkou, je možné vybrat náhodně jednu z nich. V kroku 5 se střetávají dvě cesty se stejnou délkou, takže je jen na programátorovi, či naprogramovaném algoritmu, jakou cestu vybere jako nejpravděpodobnější.

2.5 Některé konvoluční kódy

První konvoluční kódy určené k opravě shlukových chyb byly Hagelbargerův kód $(n_0, n_0 - 1)$. Další konvoluční kód se stejnou schopností korekce, který byl ovšem efektivnější a nevyžadoval tak dlouhé ochranné intervaly jako Hagelbargerův kód byl Iwadari-Maseryův kód $(m*n_0, m*k_0)$. Jako třetí byl představen Berlekamp-Preparatův kód $(n_0, n_0 - 1, m)$.

Konvoluční kódy, které slouží k detekci a opravě jednoduchých chyb se nazývají Wynerovy-Ashovy kódy $((m + 1)2^m, (m + 1)(2^m - 1))$, neboli WA-kódy. Tyto jsou svými vlastnostmi velmi blízké k Hammingovým kódům.

Všechny kódy jsou pojmenovány podle svých objevitelů.

II. PRAKTICKÁ ČÁST

3 SOFTWARE A PROGRAMOVACÍ JAZYK

3.1 Programovací jazyk C++

Jedná se o objektově orientovaný programovací jazyk, který vznikl rozšířením programovacího jazyka **C**. O jeho existenci se zasloužil dánským programátor, informatik a profesor Bjarne Stroustrup. Původní název byl „C s třídami“ (C with Classes), který později změnil Rick Mascitti na známé „C++“. Toto jméno programovací jazyk získal podle operátoru inkrementace „++“.

Programovací jazyk **C++** nepodporuje jen objektově orientované programování, ale i procedurální programování a generické programování. [10]

Objektově orientované programování (OOP) – oproti strukturovanému programování, které klade důraz na algoritmy, OPP klade důraz na data. Objektově orientované programování spočívá ve vázání dat a metod do definice třídy.

Generické programování – stejně jako OOP usiluje o vytváření jednodušších kódů a funkcí ke všeobecnému použití. K tomuto souží šablony a předlohy C++.

Programy v C++ využívají softwarové knihovny C, což jsou kolekce programových modulů, které se volají z programu. [9]

O programovacím jazyku C++ se lze také dočíst v literatuře [6] a [7].

3.2 Microsoft Visual Studio 2008

Microsoft Visual Studio 2008 (Obr. 3-1) je jedno z vývojových prostředí z řady Visual Studio od společnosti Microsoft. Visual Studio je vhodné k vytváření nejen konzolových aplikací, ale i aplikací s grafickým rozhraním.

Základní nástroje Visual Studia:

Editor kódu – podporuje automatické dokončování metod, funkcí, cyklů a kódů, kdy se objeví vyskakovací okno s nabízenými možnostmi. Nastavování záložek pro přehlednost a rychlou navigaci, ukotvení, či schování plovoucích oken a mnoho dalších.

Debugger – může pracovat se spravovaným i se strojovým kódem. Umožňuje nastavování tzv. *breakpointů* a *watte*. Breakpoint slouží k zastavení programu na daném místě během běhu programu, watte slouží ke sledování hodnot proměnných během běhu

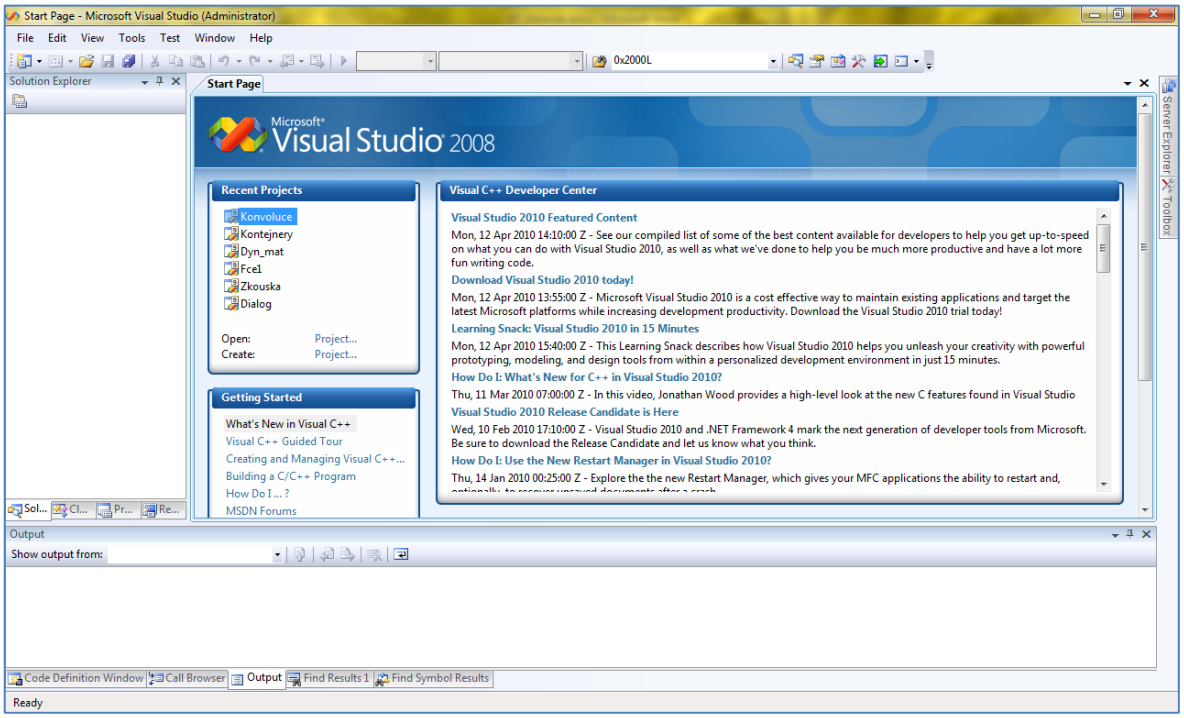
programu. *Edit and Continue* je funkce, která, jak už z názvu vyplívá, umožňuje upravovat kód během debuggingu.

Designer – vizuální designery, pomáhající s vývojem aplikací. Např.: WinForms Designer, Web designer, Designer tříd, Designer dat…

Další nástroje:

Mezi další nástroje Visual Studia patří např. průzkumník dat, průzkumník serveru, průzkumník objektů, editor vlastností…

Visual Studio umožňuje i rozšiřitelnost, a to tak, že vývojáři píši rozšíření ve formě maker, balíčků a rozšíření, které se pak „zasunou“ do Visual Studia. [11]

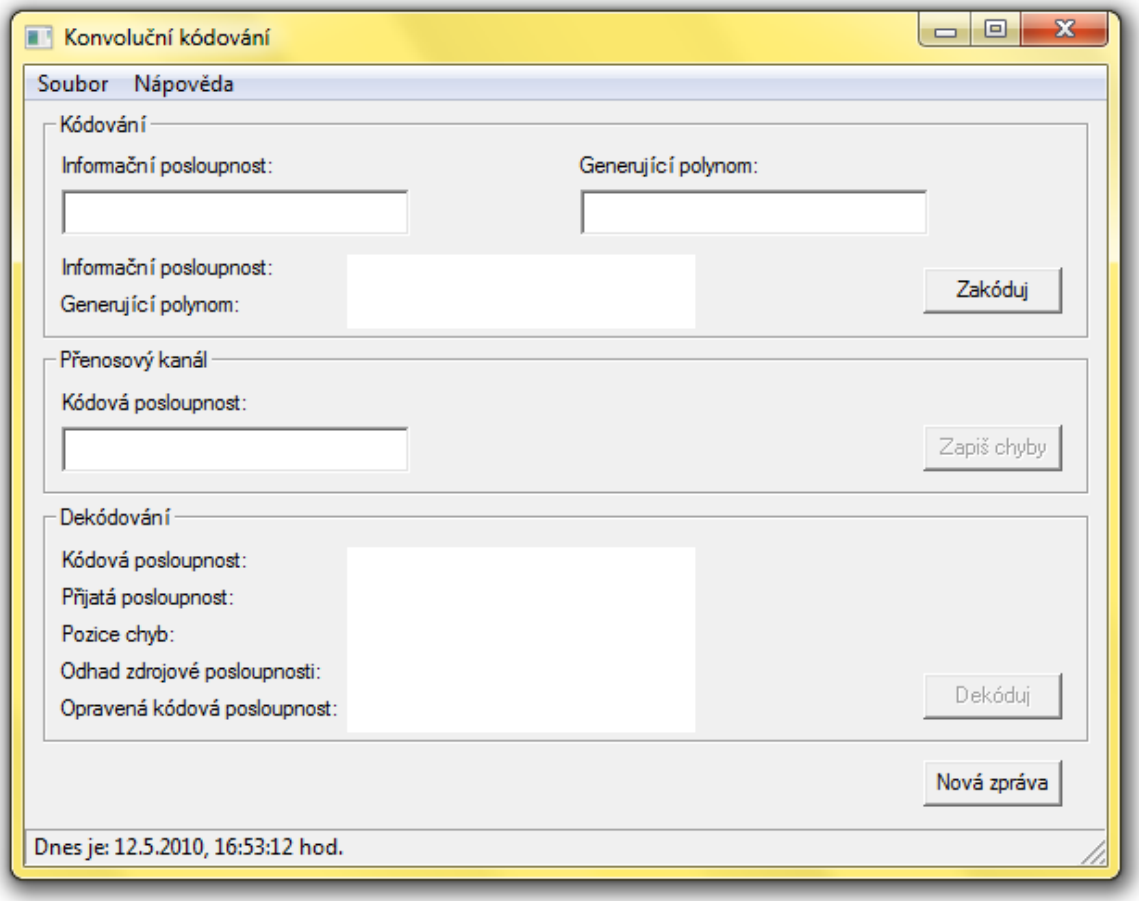


Obr. 3-1. Microsoft Visual Studio 2008

4 PROGRAM PRO SIMULACI KONVOLUČNÍHO KÓDOVÁNÍ

K praktické ukázce konvolučního kódování a dekódování vznikl program „Konvoluční kódování“ (Obr. 4-1). Program byl vytvořen za pomocí programovacího jazyka C++ a vývojového prostředí Microsoft Visual Studio 2008. K jeho tvorbě, jako učební pomůcky, sloužily literatury [6] a [7]. Ke spuštění programu slouží vytvořený exe soubor. Není třeba dalšího instalování programu, vývojového prostředí a vytváření projektu.

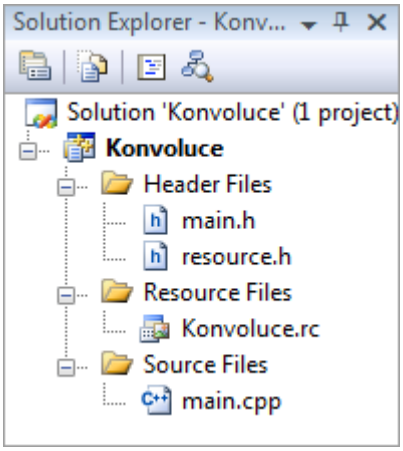
Program demonstruje kódování a dekódování konvolučního (2, 1)-kódu a je schopen detekovat a opravit až tolik chyb, kolik klopných obvodů posuvného registru kodér obsahuje. Správné opravení přijatého kódového slova a tudíž získání správné zdrojové informace závisí na pozici vzniku a počtu chyb v kódové posloupnosti. Po dekódování program zobrazí i opravenou kódovou posloupnost.



Obr. 4-1. Program simulující konvoluční kódování a dekódování

4.1 Popis zdrojového kódu

Program je vytvořený ve vývojovém prostředí Microsoft Visual Studio 2008 a obsahuje tyto soubory (Obr. 4-2):



Obr. 4-2. Soubory programu "Konvoluční kódování"

main.cpp – soubor obsahující zdrojový kód programu. Na začátku programu je includován hlavičkový soubor main.h.

main.h – hlavičkový soubor obsahující knihovny. Na začátku souboru je includován hlavičkový soubor windows.h. Je to specifický hlavičkový soubor, který obsahuje prohlášení pro funkce ve Windows API, typy dat používané různými funkcemi a podsystémy a makra používané programátory Windows. Na konci souboru je includován hlavičkový soubor resource.h.

resource.h – hlavičkový soubor obsahující definování konstant zdrojů. Zdroje mohou být: ikony, kurzory, menu, bitmapy, dialogy, klávesové zkratky a jiné.

Konvoluce.rc – kompilovaný skript zdrojů.

4.1.1 Soubor main.cpp

Soubor main.cpp je hlavní soubor programu, který obsahuje zdrojový kód. Na začátku zdrojového kódu je includován hlavičkový soubor main.h a následovně je zdrojový kód rozdělen do pěti částí.

První částí je *deklarace prvků programu*. Těmito prvky jsou hlavní okno programu, okno s nápovědou, editační pole pro zadávání hodnot, pole pro vypisování výsledků, tlačítka, rámečky oddělující části programu a status bar.

Druhá část kódu obsahuje dvě funkce, první funkce je: „`void OnWM_PAINT()`“, která obsahuje nastavení písma a určuje výpis informativního textu v hlavním okně programu. Druhá funkce: „`void OnWM_PAINT_NAPOVEDA()`“obsahuje nastavení písma a určuje výpis informativního textu v okně s nápovědou. Obě funkce jsou volány v následující části zdrojového kódu.

Třetí částí kódu je *procedura okna*. Na začátku je deklarován prostor jmen pomocí klíčových slov „`using namespace std`“, pro použití standardních funkcí. Následuje procedura: „`LRESULT CALLBACK WindowProc()`“. Na začátku procedury jsou deklarovány pomocné proměnné, ukazatel, vektory a matice. Dále již následuje zachycení zpráv od zdrojů funkcí „`switch( uMsg )`“, kde *uMsg* je identifikátor zpráv:

- `case WM_CLOSE:` Ukončení programu volané od křížku (pravý horní roh aplikace) nebo od volby „Ukončit“ (Obr. 4-8). Zde se program ptá, zda si uživatel opravdu přeje program ukončit. Pokud ano, zpráva volá „`case WM_DESTROY`“.
- `case WM_COMMAND:` Zachycení uživatelských zpráv od tlačítek, menu a editačních polí:

  a) funkcí „`switch( LOWORD(wParam) )`“, kde *LOWORD(wParam)* je ID nabídky nebo ovládacího prvku:
     - `case ID_ABOUT:` Zobrazení informací o programu od volby „Zobrazit nápovědu“ (Obr. 4-3).
     - `case ID_END:` Ukončení programu od od volby „Ukončit“ (Obr. 4-8). Zpráva volá „`case WM_CLOSE`“.
     - `case ID_NAPOVEDA:` Zobrazení nápovědy k programu od volby „O programu Konvoluční kódování“ (Obr. 4-3).
     - `case IDC_KONTROLA:` Kontrola obsahu editačních polí. Kontrola je volána od editačních polí pro zadávání informační posloupnosti, denerujícího polynomu i editace kódové posloupnosti.
     - `case IDC_KODUJ:` Zpráva od volby „Zakóduj“ (Obr. 4-8). Nastaví *lParam* na *g_ButtonKoduj*.

- `case IDC_CHYBY:` Zpráva od volby „Zapiš chyby“ (Obr. 4-8). Nastaví *lParam* na *g_ButtonChyby*.
- `case IDC_DEKODUJ:` Zpráva od volby „Dekóduj“ (Obr. 4-8). Nastaví *lParam* na *g_ButtonDekoduj*.
- `case IDC_NOVA:` Zpráva od volby „Nová zpráva“ (Obr. 4-8). Nastaví *lParam* na *g_ButtonNova*.

b) testováním parametru *lParam*, což je handle „ukazatel“ prvku okna:

- `(LPARAM)g_ButtonKoduj:` Bylo zmáčknuto tlačítko „Zakóduj“ v hlavním okně (Obr. 4-4), nebo volba menu „Zakóduj“. Zde se provádí test obsahu editačních polí pro zdrojovou informaci a generující mnohočlen a následně se provede zakódování zdrojové informace do kódové posloupnosti.
- `(LPARAM)g_ButtonChyby:` Bylo zmáčknuto tlačítko „Zapiš chyby“ v hlavním okně (Obr. 4-9), nebo volba menu „Zapiš chyby“. Zde se provádí test obsahu editačního pole pro úpravu kódové posloupnosti a následně se provede zapsání chyb do kódové posloupnosti.
- `(LPARAM)g_ButtonDekoduj:` Bylo zmáčknuto tlačítko „Dekóduj“ v hlavním okně (Obr. 4-12), nebo volba menu „Dekóduj“. Zde se provádí dekódování kódové posloupnosti a vypsání výsledků do okna programu.
- `(LPARAM)g_ButtonNova:` Bylo zmáčknuto tlačítko „Nová zpráva“ v hlavním okně (Obr. 4-1), nebo volba menu „Nová zpráva“. Zde se provádí smazání obsahů všech editačních polí a výsledných výpisu do okna programu.
- `(LPARAM)g_ButtonZavri:` Bylo zmáčknuto tlačítko „Zavři“ v okně s nápoděvou k programu. Provede se zavření okna s nápovědou.

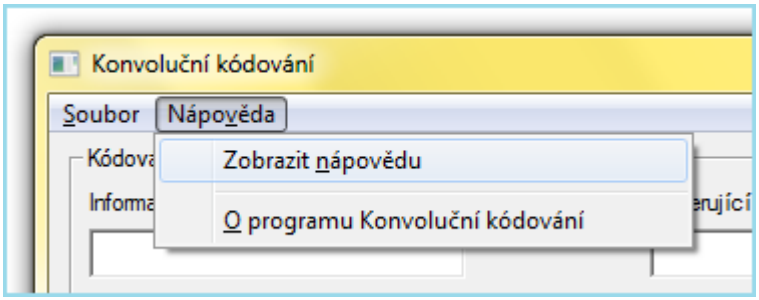
- `case WM_DESTROY:` Ukončení běhu programu.
- `case WM_SIZE:` Změna velikosti status baru při změně velikosti okna.
- `case WM_TIMER:` Nastavení a zobrazení textu ve status baru.
- `case WM_GETMINMAXINFO:` Nastavení pevné velikost okna.
- `case WM_PAINT:` Volání funkcí z druhé části kódu pro vykreslení textů v okně.

Následující, čtvrtou částí kódu, je *registrace třídy okna a vytvoření třídy okna*: „`BOOL Inicializace()`“. V této části se provádí registrace třídy okna a vytváření prvků programu. Těmito prvky jsou hlavní okno programu, okno s nápovědou, editační pole pro zadávání hodnot, pole pro vypisování výsledků, tlačítka, rámečky oddělující části programu a status bar. Na konci této části je nastavení písma textů prvků a funkce „`ShowWindow(g_hWnd, SW_SHOW)`“ zajistí zobrazení hlavného okna po spuštění programu.

Pátou a poslední částí zdrojového kódu je *hlavní funkce programu pro Windows*: „`int WINAPI _tWinMain()`“. Zde probíhá test na inicializaci prvků programu (čtvrtá část kódu), vznikne-li chyba, vrátí test hodnotu -1 a program se ukončí. Je-li inicializace bechybná, následuje *hlavní smyčka zpráv,* která zajišťuje běh programu.

4.2 Obsluha programu

Program se skládá z tří částí, z nichž každá je označená rámečkem a příslušným popisem. Stručný popis programu a jeho ovládání je i přímo v programu a lze jej zobrazit přes menu: Nápověda → Zobrazit nápovědu (Obr. 4-3). Nápověda se ukončí stisknutím tlačítka „Zavři“.

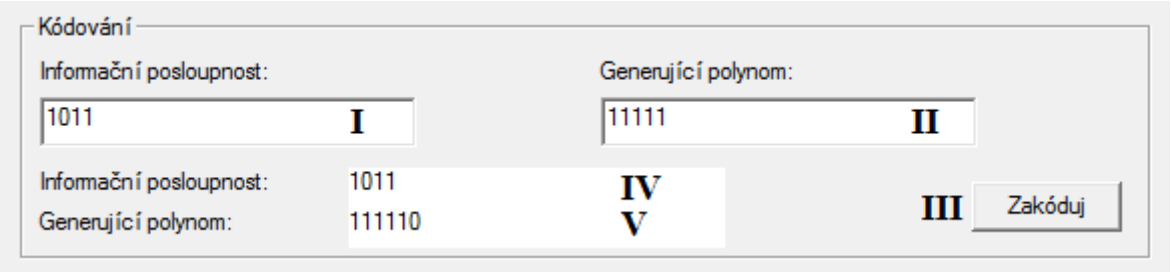


Obr. 4-3. Nápověda k programu

Volbou: Nápověda → O programu Konvoluční kódování, se zobrazí informace o programu.

4.2.1 Kódování

První částí programu je kódování informační posloupnosti (Obr. 4-4).



Obr. 4-4. První část programu – Kódování

Popis ovládacího panelu:

I – Editační pole, do něhož se zapisuje informace, kterou chceme zakódovat.

II – Editační pole, do kterého se zapisuje generující polynom. Generující polynom udává, jak bude vypadat konvoluční kodér a kolik bude obsahovat klopných obvodů posuvného registru.

III – Tlačítko „Zakóduj“, po jeho zmáčknutí dojde k zakódování zdrojové informace. Tlačítko je v základním stavu aktivováno.

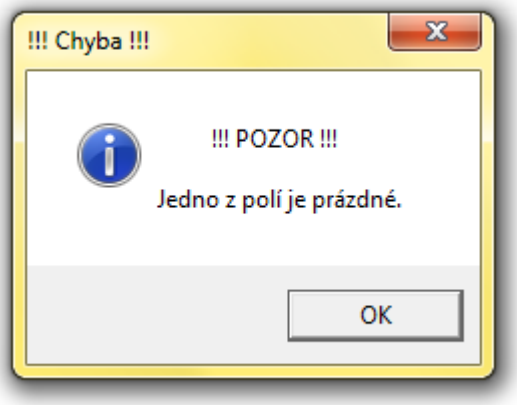
IV – Kontrolní výpis informační posloupnosti. Po zmáčknutí tlačítka „Zakóduj“ se zde vypíše zadaná informační posloupnost.

V – Kontrolní výpis generujícího polynomu. Po zmáčknutí tlačítka „Zakóduj“ se zde vypíše zadaný generující polynom.

Postup při kódování zdrojové informace:

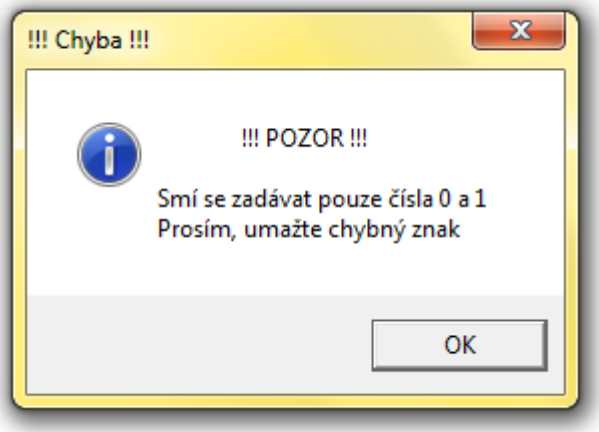
Jako první se zapíše zdrojová informace do editačního pole „Informační posloupnost:“ a generující polynom do editačního pole „Generující polynom:“. Poté se stisknutím tlačítka „Zakóduj“ provede zakódování vstupní informace. Po stlačení tlačítka může dojít ke čtyřem situacím:

- **Chyba – Prázdné editační pole:** je-li po stisknutí tlačítka „Zakóduj“ jedno, případně obě editační pole prázdné, vyběhne dialog s upozorněním (Obr. 4-5). Program bude pokračovat dále, až když budou obě pole naplněna hodnotami.



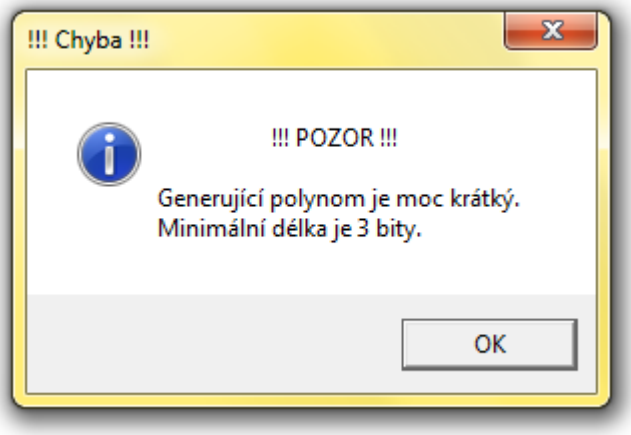
Obr. 4-5. Chyba – Prázdné editační pole

- **Chyba – Chybný znak:** obsahuje-li jedno, případně obě editační pole jinou hodnotu, než je binární, tj. 0 a 1, vyběhne dialog s upozorněním (Obr. 4-6). Program bude pokračovat dále, až když budou obě pole obsahovat pouze binární hodnoty. Jiné znaky než numerické jsou mazány automaticky, čísla 2 – 9 je nutno mazat manuálně.



Obr. 4-6. Chyba – Chybný znak

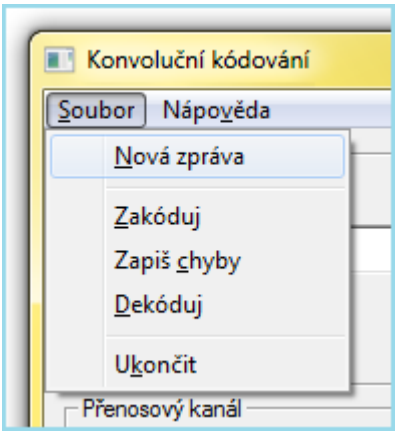
- **Chyba – Minimální délka:** je-li generující polynom menší, než 3 znaky, vyběhne dialog s upozorněním (Obr. 4-7). Program pokračuje dále, až když je délka generujícího polynomu rovna nebo vetší jak 3 znaky. Nejmenší povolené hodnoty generujícího polynomu jsou: 001, 011, 111 a 101.



Obr. 4-7. Chyba – Minimální délka

- **Vše v pořádku:** obsahují-li obě pole binární hodnoty a délka generujícího polynomu je rovna nebo větší jak 3 znaky, je vše v pořádku a program pokračuje dále. Pro kontrolu se vypíše informační posloupnost a generující polynom, body IV a V (Obr. 4-3).

Pro zakódování informační posloupnosti může být použito stisknutí tlačítka „Zakóduj“, nebo volby přes menu: Soubor → Zakóduj (Obr. 4-8), která je v základním stavu aktivní.



Obr. 4-8. Ovládání přes menu

4.2.2 Přenosový kanál

Druhou částí programu je přenosový kanál (Obr. 4-9), který simuluje přenos kódové posloupnosti přes přenosový kanál a zavedení chyb do kódové posloupnosti.

Obr. 4-9. Druhá část programu – Přenosový kanál

Popis ovládacího panelu:

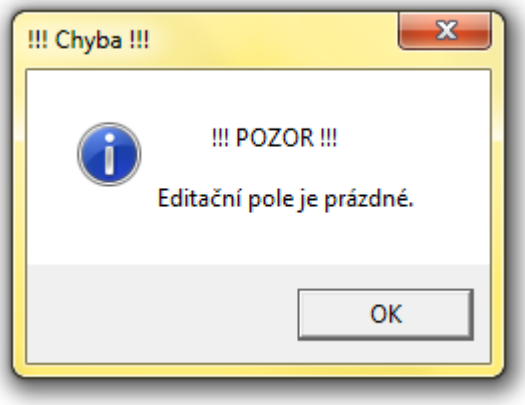
I – Editační pole, ve kterém se upravuje kódová posloupnost, vznikají chyby.

II – Tlačítko „Zapiš chyby“, po jeho zmáčknutí dojde k zápisu chyb do kódové posloupnosti. Tlačítko je v základním stavu deaktivováno.

Postup při přenosu kódové posloupnosti:

Proběhne-li proces kódování v pořádku, je do pole „Kódová posloupnost:“ vepsána kódová posloupnost z výstupu kodéru a tlačítko „Zapiš chyby“ se stane aktivním. V této fázi je stále možné měnit zdrojovou informaci nebo generující polynom, ovšem poté je nutné znovu zakódovat informaci stisknutím tlačítka „Zakóduj“. V poli „Kódová posloupnost:“ je potom zobrazena nová kódová posloupnost. Kódovou posloupnost je možno měnit, tzn. simulovat vznikání chyb při přenosu kódové posloupnosti. Poté se stisknutím tlačítka „Zapiš chyby“ provede zápis chyb do kódové posloupnosti. Po stlačení tlačítka může opět dojít ke čtyřem situacím:

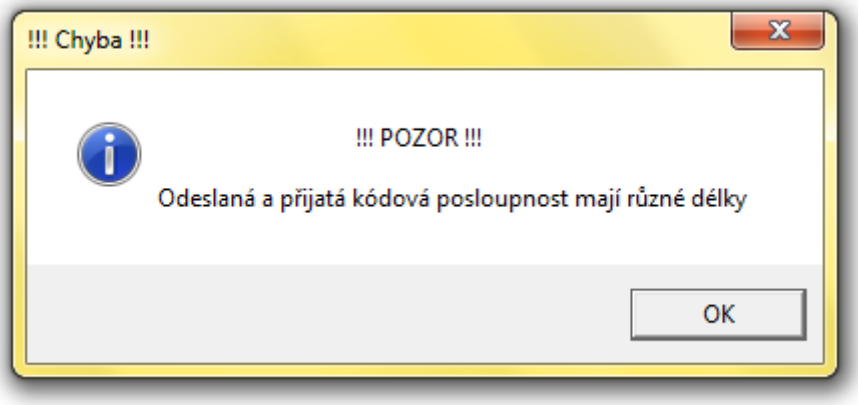
- **Chyba – Prázdné editační pole:** stejná situace, jako u kódování. Program bude pokračovat dále, až když bude pole naplněno binární hodnotou. Je-li po stisknutí tlačítka „Zapiš chyby“ editační pole prázdné, vyběhne dialog s upozorněním (Obr. 4-10).



Obr. 4-10. Chyba – Prázdné editační pole

- **Chyba – Chybný znak:** stejná situace, jako u kódování. Obsahuje-li editační pole jinou hodnotu, než je binární, tj. 0 a 1, vyběhne dialog s upozorněním (Obr. 4-6). Program bude pokračovat dále, až když bude pole obsahovat pouze binární hodnoty. Jiné znaky než numerické jsou mazány automaticky, čísla 2 – 9 je nutno mazat manuálně.

- **Chyba – Různé délky:** obdobná situace, jako u kódování. Mají-li odeslaná kódová posloupnost a přijatá kódová posloupnost různé délky, vyběhne dialog s upozorněním (Obr. 4-11). Program pokračuje dále, až když mají odeslaná kódová posloupnost a přijatá kódová posloupnost stejné délky.



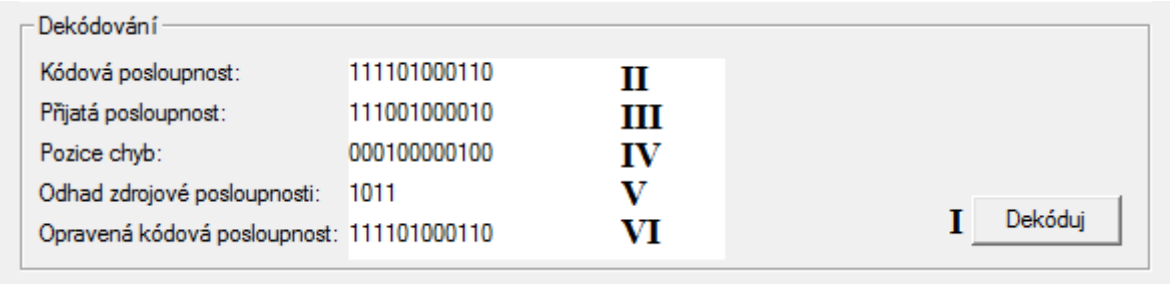
Obr. 4-11. Chyba – Různé délky posloupností

- **Vše v pořádku:** stejná situace, jako u kódování. Obsahuje-li pole binární hodnoty a délka přijaté posloupnosti je rovna délce odeslané posloupnosti, je vše v pořádku a program pokračuje dále.

Pro zapsání chyb do kódové posloupnosti může být použito stisknutí tlačítka „Zapiš chyby“, nebo volby přes menu: Soubor → Zapiš chyby (Obr. 4-8), která je v základním stavu neaktivní, aktivuje se až po úspěšném zakódování informace.

4.2.3 Dekódování

Třetí částí programu je dekódování kódové posloupnosti a získání nejpravděpodobnějšího odhadu zdrojové informace (Obr. 4-12).



Obr. 4-12. Třetí část programu – Dekódování

Popis ovládacího panelu:

I – Tlačítko „Dekóduj“, po jeho zmáčknutí dojde k dekódování kódové posloupnosti. Tlačítko je v základním stavu deaktivováno.

II – Po zmáčknutí tlačítka „Dekóduj“ se zde vypíše odeslaná kódová posloupnost z výstupu kodéru.

III – Po zmáčknutí tlačítka „Dekóduj“ se zde vypíše přijatá kódová posloupnost na vstupu dekodéru.

IV – Po zmáčknutí tlačítka „Dekóduj“ se zde pro lepší orientaci vypíše pozice chyb vzniklých v kódové posloupnosti při přenosu komunikačním kanálem. Pozice chyby je vyznačena jedničkou.

V – Po zmáčknutí tlačítka „Dekóduj“ se zde vypíše nejpravděpodobnější odhad původní zdrojové informace získaný dekódováním kódové posloupnosti.

VI – Po zmáčknutí tlačítka „Dekóduj“ se zde vypíše opravená kódová posloupnost tak, jak ji opraví dekodér.

Postup při dekódování kódové posloupnosti:

Proběhne-li proces zanesení chyb do kódové posloupnosti v pořádku, stane se tlačítko „Dekóduj“ aktivním a tlačítko „Zakóduj“ je deaktivováno. Po stisknutí tlačítka „Dekóduj“ se vypíší všechny informace o dekódování, body II - VI. V této fázi je stále možné měnit kódovou posloupnost. Uživatel si tak může vyzkoušet zanesení různého počtu chyb na různé pozice do kódové posloupnosti, případně zkusit bezchybný přenos posloupnosti. Pro informaci o bezchybné kódové posloupnosti a pozic chyb v předešlém příkladu slouží výpis informací po dekódování. Ovšem poté je nutné znovu zapsat chyby do kódové posloupnosti stisknutím tlačítka „Zapiš chyby“ a následným stisknutím tlačítka „Dekóduj“ dekódovat kódovou posloupnost.

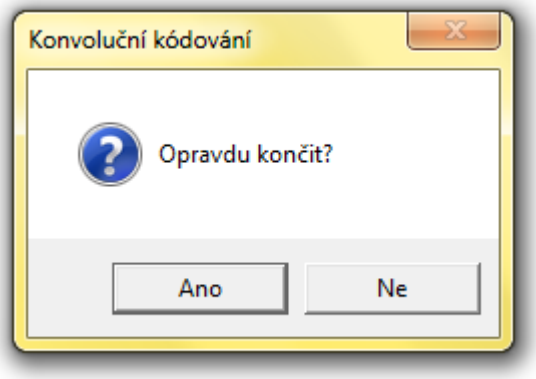
Pro dekódování kódové posloupnosti může být použito stisknutí tlačítka „Dekóduj“, nebo volby přes menu: Soubor → Dekóduj (Obr. 4-8), která je v základním stavu neaktivní, aktivuje se až po úspěšném vložení chyb do kódové posloupnosti.

4.2.4 Nová zpráva

K vyzkoušení kódování a dekódování jiného konvolučního (2, 1)-kódu slouží tlačítko „Nová zpráva“ (Obr. 4-1)., nebo volba přes menu: Soubor → Nová zpráva (Obr. 4-8). Při zvolení nové zprávy je tlačítko „Zakóduj“ opět aktivováno a tlačítka „Zapiš chyby“ a „Dekóduj“ deaktivována (Obr. 4-1).

4.2.5 Konec

Pro ukončení programu lze využít klasického zrušení okna křížkem (pravý horní roh), nebo volbou přes menu: Soubor → Ukončit (Obr. 4-8). Před úplným ukončením programu se program zeptá, zda si je uživatel ukončením jistý nebo ne (Obr. 4-13). Pokud zvolí NE, vrátíte se zpět do programu, pokud zvolí ANO, je program ukončen.



Obr. 4-13. Ukončení programu

ZÁVĚR

Cílem bakalářské práce bylo seznámení s konvolučními kódy a naprogramování algoritmu, jenž by demonstroval kódování a dekódování konvolučního kódu.

V teoretické části bakalářské práce jsem se zabýval obecným úvodem do kódování. Po obecném úvodu jsem se zabýval problematikou konvolučních kódů, a to od sestavení kodéru, kódování, přenos kódového slova reálným komunikačním kanálem až po dekódování konvolučních kódů. V této části jsem se zaměřil na konvoluční (2, 1)-kódy, tzn., že 1-bitový znak zdrojové informace na vstupu je transformován do 2-bitových znaků kódové posloupnosti na výstupu. Při kódování konvolučního kódu jsem se zabýval matematickou metodou, $v(x) = f(x) * G$, kde $f(x)$ je zdrojová informace, G je generující matice a $v(x)$ je výsledná kódová posloupnost, a následně i známější metodou, která spočívá v grafickém znázornění kódování, tzv. mřížkový diagram. Při dekódování kódové posloupnosti jsem se zaměřil na Viterbiho algoritmus.

V druhé, praktické části jsem popsal ovládání programu, který byl vytvořen pro simulaci konvolučního kódování a dekódování. Pro zakódování vstupní informace jsem využil matematického modelu, $v(x) = f(x) * G$. Pro dekódování přijaté kódové posloupnosti jsem využil znalostí Viterbiho algoritmu. Vytvořil jsem tabulkovou podobu mřížkového diagramu a pomocí Viterbiho algoritmu jsem do ní ukládal Hammingovi vzdálenosti. Nakonec jsem zpětným průchodem tabulkou určil nejpravděpodobnější podobu původní zdrojové informace. Naprogramovaný algoritmus je schopen opravit až tolik chyb, kolik klopných obvodů posuvného registru kodér obsahuje. K vytvoření programu „Konvoluční kódování“ byl použit programovací jazyk C++ a vývojové prostředí Microsoft Visual Studio 2008.

Program „Konvoluční kódování“ je jen základní zpracování okna, editačních polí a tlačítek pro simulaci konvolučního kódování. Program by mohl být dále rozšířen například o grafické znázornění podoby kodéru či grafickou podobu mřížkového diagramu. Takto rozšířený program by již mohl sloužit, jako studijní pomůcka pro názorné ukázky tvorby kodéru, postupu dekódování, či sestavení a průchod mřížkovým diagramem.

CONCLUSION

The aim of this bachelor work was to acquaint with the convolution codes and programming algorithm, which would illustrate the encoding and decoding of convolution code.

I focused on a general introduction to coding in the theoretical part of the bachelor work. After a general introduction, I dealt with problems of convolution codes, namely the encoder assembly, encoding, transmission of the codeword over the real communication channel and decoding of convolution codes. I focus on the convolution (2, 1)-codes in this section, i.e. the 1-bit character input source information are transformed into 2-bit character code sequence output. I dealt with mathematical methods for coding convolution code, $v(x) = f(x) * G$, where $f(x)$ is the source information, G is generating matrix and $v(x)$ is the resulting code sequence, and subsequently the better-known method which is the graphic representation of coding, known as trellis diagram. I focused on the Viterbi algorithm when I was decoding the code sequence.

In the second part I described the control program, which was created to simulate convolution encoding and decoding. I used a mathematical model to encode the input information, $v(x) = f(x) * G$. The knowledge of Viterbi algorithm was used to decode received code sequence. I created a spreadsheet from trellis diagram and using the Viterbi algorithm I saved the Hamming distance in it. Finally, I went back through the table to determine the most likely form of the original source information. Programmed algorithm is able to correct as many errors as many flip-flop circuits of a shift register are included in encoder. The C++ programming language and the Microsoft Visual Studio 2008 were used to create the "Convolution coding" program.

Program "Convolution coding" is a basic window with edit boxes and buttons to simulate the convolution coding. The program could be enhanced by, for example, a graphical representation of the encoder design or graphic design of the trellis diagram. This extended program would be able to serve as a learning tool for the demonstration encoder assembly, decoding process or the preparation and passage through the trellis diagram.

SEZNAM POUŽITÉ LITERATURY

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

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

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

[4] Kocourek, P., Novák, J.: Přenos informace. ČVUT, Praha, 2004, ISBN 8001028925

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

[6] Liberty, Jesse.: Naučte se C++ za 21 dní. Vydání první, Computer Press, Praha, 2002, 766 s, ISBN 80-7226-774-4.

[7] Chalupa, Radek.: 1001 tipů a triků pro Visual C++. Vydání první, Computer Press, Brno, 2003, 434 s, ISBN 80-7226-842-2.

[8] *Wikipedie : Otevřená encyklopedie* [online]. 21. 6. 2006, 19. 1. 2010 [cit. 2010-03-17]. Konvoluce. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Konvoluce>.

[9] CHURÝ, Lukáš. *Programujte : Zaměřeno na informační technologie* [online]. 12. 04. 2005, 3. 2. 2008 13:35 Aktualizováno [cit. 2010-05-12]. C++ - Úvod. Dostupné z WWW: <http://programujte.com/?akce=clanek&cl=2005080601-c++-uvod>. ISSN 1801-1586.

[10] *Wikipedie : Otevřená encyklopedie* [online]. 20. 6. 2004, Stránka byla naposledy editována 6. 4. 2010 [cit. 2010-05-12]. C++. Dostupné z WWW: <http://cs.wikipedia.org/wiki/C%2B%2B>.

[11] *Wikipedie : Otevřená encyklopedie* [online]. 21. 3. 2006, Stránka byla naposledy editována 28. 4. 2010 [cit. 2010-05-12]. Microsoft Visual Studio. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Visual_Studio>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

$\oplus$	sečítání modulo 2
$\otimes$	násobení modulo 2
0	nula, znak binární abecedy
1	jednička, znak binární abecedy
API	(Application Programming Interface) rozhraní pro programování aplikací
BCH kódy	Bose-Chaudhuriho-Hocquenghemovy kódy
C, C++	programovací jazyky
d(A, B)	Hammingova vzdálenost
d_{min}	min Ham vzd
f(x)	zdrojová posloupnost
G	generující matice
g(x)	generující mnohočlen
i, j	označení času
K	označení konvolučního kódu, omezující délka konvolučního kódu
k	počet vstupů konvolučního kodéru
L	délka zdrojové posloupnosti
LOWORD(wParam)	ID nabídky nebo ovládacího prvku
lParam	handle prvku okna
n	počet výstupů konvolučního kodéru
(n, k)-kód	typ konvolučního kódu
OOP	Objektově orientované programování
uMsg	identifikátor zpráv
v(x)	výstupní kódová osloupnost
WA-kódy	Wynerovy-Ashovy kódy
wParam	handle kontextu zařízení

SEZNAM OBRÁZKŮ

SEZNAM TABULEK