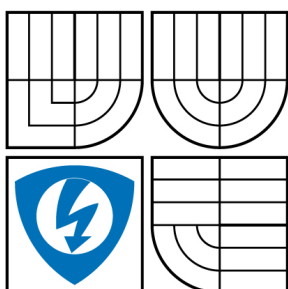


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ALGORITMY PRO DEKÓDOVÁNÍ REED-SOLOMONOVA PROTICHYBOVÉHO KÓDU

ALGORITHMS FOR DECODING THE REED-SOLOMON ERROR CONTROL CODE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

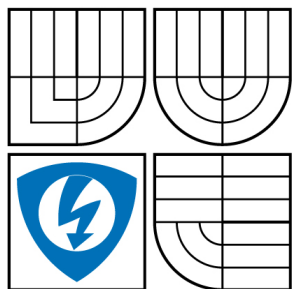
AUTOR PRÁCE
AUTHOR

Bc. TOMÁŠ TIEFTRUNK

VEDOUcí PRÁCE
SUPERVISOR

Ing. PAVEL ŠILHAVÝ, Ph.D.

BRNO 2008



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Tieftrunk Tomáš Bc.
Ročník: 2

ID: 89201
Akademický rok: 2007/2008

NÁZEV TÉMATU:

Algoritmy pro dekódování Reed-Solomonova protichybového kódu

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku protichybového zabezpečení pomocí Reed-Solomonova kódu. Dále se zaměřte na problematiku dekódování zprávy zabezpečené pomocí Reed-Solomonova kódu. V programovém prostředí Matlab vytvořte demonstrační program, který bude možno využít jako výukovou pomůcku. Použitelnost testovaného algoritmu ověřte implementací protichybového zabezpečení datové komunikace, prostřednictvím RS232 rozhraní, mezi počítačem a jednočipovým mikrokontrolérem. Vhodnou formou otestujte spolehlivost datového přenosu.

DOPORUČENÁ LITERATURA:

- [1] Morelos-Zaragoza, R.. The Art of Error Correcting Coding. 2005. John Wiley & Sons Ltd., ISBN: 0-470-44782-4.
- [2] Lin, S., Costello, D. J.. Error Control Coding: Fundamentals and Applications, second edition, Prentice Hall: Englewood Cliffs, NJ, 2005 ISBN: 0-13-042672-5.
- [3] Skalar, B.. Digital Communications, Fundamentals and applications, Prentice-Hall, 2003, ISBN 0-13-084788-7.

Termín zadání: 11.2.2008

Termín odevzdání: 28.5.2008

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

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

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

LICENČNÍ SMLOUVA

POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. Tomáš Tiefertunk
Bytem: Sv. Čecha 1074, 73581, Bohumín - Nový Bohumín
Narozen/a (datum a místo): 17.7.1983, Bohumín

(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 244/53, 60200 Brno 2
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Ing. Kamil Vrba, CSc.

(dále jen "nabyvatel")

Článek 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

- ☐ disertační práce
- ☒ diplomová práce
- ☐ bakalářská práce

jiná práce, jejíž druh je specifikován jako

(dále jen VŠKP nebo dílo)

Název VŠKP: Algoritmy pro dekódování Reed-Solomonova protichybového kódu

Vedoucí/školicitel VŠKP: Ing. Pavel Šilhavý, Ph.D.

Ústav: Ústav telekomunikací

Datum obhajoby VŠKP:

VŠKP odevzdal autor nabyvateli v:

- ☒ tištěné formě - počet exemplářů 1
- ☒ elektronické formě - počet exemplářů 1

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užívat, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
 - ☒ ihned po uzavření této smlouvy
 - ☐ 1 rok po uzavření této smlouvy
 - ☐ 3 roky po uzavření této smlouvy
 - ☐ 5 let po uzavření této smlouvy
 - ☐ 10 let po uzavření této smlouvy(z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísní a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne:

.....

Nabyvatel

.....

Autor

ABSTRAKT

Abstrakt práce v češtině

Práce pojednává o problematice zabezpečení dat proti výskytu chyb Reed Solomonovým kódem. Jedná se o blokový, cyklický, systematický kód, pracující se znaky. Proces dekódování, tedy opravování chyb, je výpočetně náročný. V práci je podrobně popsán Berlekamp-Maseyho algoritmus, používaný pro nalezení chybového polynomu. Proces je ilustrován pomocí palikace v prostředí Matlab. Praktická implementace kódu je v zabezpečení přenosu dat přes rozhraní RS232. Zabezpečená komunikace probíhá mezi počítačem a mikroprocesorem.

KLÍČOVÁ SLOVA

Klíčová slova v češtině

Reed Solomon, FEC, Berlekamp-Masey, zabezpečený přenos, RS232

ABSTRACT

Abstract in English

Thesis discuss about effort to ensure from error, which may occur during transmission over noisy channel. There's used Reed Solomon code. It's block, cyclic and systematic code, which is symbol orientated. Computational process of decoding is mathematically time-consuming. In thesis is closely described Berlekamp-Masey algorithm, used in decoding to evaluate error polynomial. Process is illustrated in application in Matlab. Practical realization uses Reed Solomon code in communication over RS232. Communication is established between computer and microcomputer.

KEYWORDS

Keywords in English

Reed Solomon, FEC, Berlekamp-Masey, error correction, RS232

TIEFTRUNK, T. Algoritmy pro dekódování Reed-Solomonova protichybového kódu. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 50 s. Vedoucí diplomové práce Ing. Pavel Šilhavý, Ph.D.

Prohlášení

Prohlašuji, že svůj semestrální projekt na téma Algoritmy pro dekódování Reed-Solomonova protichybového kódu jsem vypracoval samostatně pod vedením vedoucího semestrálního projektu a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne

.....

(podpis autora)

Poděkování

Děkuji vedoucímu diplomové práce Ing. Pavlu Šilhavému, Ph.D. , za užitečnou pomoc a cenné rady při vypracování této diplomové práce.

Obsah

Úvod.....	12
1. Rozdělení kódů	13
2. Reed Solomonovy kódy	14
2.1 RS kód.....	14
2.1.1 Galoisovo těleso.....	15
2.2 RS dekodér.....	16
2.2.1 Výpočet syndromů.....	17
2.2.2 Stanovení chybového polynomu - Berlekamp – Masey algoritmus	17
2.2.3 Pozice chyb – Chienovo vyhledávání.....	21
2.2.4 Velikost chyb – Forneyův algoritmus.....	21
2.2.5 Oprava chyb	21
3. Prokládání	22
3.1 Blokové prokládání.....	22
3.2 Konvoluční prokládání.....	23
4. Program v prostředí Matlab	24
5. Zabezpečení komunikace přes RS232	26
5.1 Popis rozhraní RS232	26
5.2 Použitý zabezpečovací kód	29
5.3 Struktura použitého protokolu	29
5.3.1 Struktura použitého protokolu při prokládání.....	32
5.4 Program pro PC.....	33
5.4.1 Samotná funkce programu	36
5.5 Program pro mikropočítač	39
5.5.1 Samotná funkce programu mikropočítače	41
5.6 Porovnání jednotlivých kódů	42
6. Závěr	46
Použitá literatura	47
Seznam použitých zkratk	48
Přílohy.....	50

Seznam obrázků

Obr. č. 1: Struktura zprávy RS kódu.....	15
Obr. č. 2: Postup dekódování RS kódu.....	16
Obr. č. 3: Vývojový diagram Berlekamp – Maseyho algoritmu.....	18
Obr. č. 4: Princip blokového prokládání, schéma rozprostření chyb.....	23
Obr. č. 5: Aplikace v prostředí Matlab.....	24
Obr. č. 6: Konektor CANNON 9 rozhraní RS232.....	26
Obr. č. 7: Struktura přenášených dat.....	27
Obr. č. 8: Napěťové úrovně detekce logických úrovní RS232.....	28
Obr. č. 9: Struktura přenášeného rámce.....	30
Obr. č. 10: Struktura záhlaví.....	30
Obr. č. 11: Struktura prokládaného rámce při použití kódu RS 15/13.....	32
Obr. č. 12: Hlavní okno programu rs.exe , výběr COM portu.....	33
Obr. č. 13: Okno zobrazující průběh přenosu dat.....	34
Obr. č. 14: Okno s výsledky testování chybovosti linky.....	35
Obr. č. 15: Vývojový diagram – start programu.....	36
Obr. č. 16: Vývojový diagram – odesílání dat.....	37
Obr. č. 17: Vývojový diagram – testování chybovosti linky.....	38
Obr. č. 18: Kit STK 500.....	39
Obr. č. 19: Zapojení mikropočítače do systému.....	40
Obr. č. 20: Vývojový diagram programu v mikropočítači.....	41
Obr. č. 21: Graf zobrazující dobu přenosu jednotlivých souborů použitými kódy.....	43
Obr. č. 22: Znázornění dob přenosů pro přenesení stejného množství dat.....	44

Seznam tabulek

Tab. č. 1: Funkce jednotlivých pinů rozhraní RS232.....	26
Tab. č. 2: Přenosové rychlosti.....	29
Tab. č. 3: Použité kódy a jejich vlastnosti.....	29
Tab. č. 4: Význam jednotlivých bitů v záhlaví.....	31
Tab. č. 5: Přenosové rychlosti dat při použití kódů z tab. č. 3.....	32
Tab. č. 6: Výsledky testování přenosem souborů.....	42

Úvod

Při práci s daty může se může stát, že dojde k jejich poškození, nebo ztrátě. A to buď vlivem okolního rušení během přenosu a nebo chybovostí média, na které data ukládáme. Z tohoto důvodu zavádíme různé algoritmy, pomocí kterých se pokoušíme zabránit jejich vzniku, a nebo je alespoň eliminovat. Procesu, který proti vzniklým chybám chrání, se říká zabezpečení protichybovým kódem. Díky přidaným informacím k datům, které zabezpečujeme, jsme schopni do určité míry vzniklé chyby opravit nebo alespoň detekovat. Jestliže je použit detekční kód, chybu pouze detekuje. Oprava se takovém případě řeší opětovným vysíláním dat. Takovýto systém se užívá například v Ethernetu. Naopak při užití korekčního kódu umíme vzniklé chyby opravit. Teprve až při výskytu neopravitelné chyby, dochází k opětovnému vysílání. Chyby, které vznikají při přenosu, můžeme rozdělit na náhodné a shlukové. Chyby se vyskytují zcela náhodně, ale shlukové se vyskytují vždy ve větším množství za sebou (shluky). Podle typu přenosu a předpokládaného rušení si vybereme vhodný kód. Vždy se tak rozhodujeme mezi vyšší přenosovou rychlostí dat, a nebo naopak jejich bezchybností. Reed Solomonovy kódy jsou korekční, blokové, cyklické. Dokážou opravit velké množství chyb, a proto své využití nacházejí především v CD a pevných discích a v přenosových systémech xDSL i DVB.

1. Rozdělení kódů

Podle přenosového média a místem přenosu můžeme určit, jaké chyby budou na médiu vznikat. Pak zvolíme kód, který bude pro přenos nejefektivnější. Kódů existuje celá řada, a neexistuje jednotné rozdělení. Lze je ale rozdělit podle jejich charakteristických vlastností.

Blokové a stromové kódy

Blokové kódy pracují vždy s určitým blokem dat, který zabezpečí. Oproti tomu stromové kódy pracují i nejenom s aktuálně přijatým blokem dat, ale také s daty přijatými již v předchozích krocích.

Detekční a korekční kódy

Pomocí detekčních kódů jsme pouze schopni odhalit vzniklou chybu, ale nejsme schopni ji následně opravit. Toto kódování se používá hlavně v přenosových médiích s nízkou chybovostí. V případě že se chyba vyskytla, považují se data za špatná a zahodí se. Následně se vyšle požadavek na opakování vyslaných dat. Nejznámějším detekčním kódem je CRC používaný v Ethernetu. U korekčních kódů jsme z přidaných informací data schopni do jisté míry opravit

Systematické a nesystematické

U systematického kódu se ve výstupním slově dají oddělit od sebe zabezpečovací a informační bity. Kdežto u nesystematických to nelze, protože výstupní slovo se odvodí od poloh jedniček a nul ve vstupním slově.

Lineární a ostatní

U lineárních kódů jsou pomocné (zabezpečující) informace zařazeny za nebo před přenášená data. Takže lze od sebe oddělit zabezpečovací a informační část. Ale u ostatních lineárních kódů jsou pomocné informace zařazeny mezi přenášená data. Podle parametrů kódu lze zjistit polohu těchto dat.

2. Reed Solomonovy kódy

Reed-Solomonovy kódy (dále jen RS) spadají do kategorie korekčních blokových cyklických systematických lineárních kódů. Řadíme je do třídy BCH kódů. RS kódy jsou symbolově orientované, což znamená že pracují s celými symboly, a ne s jednotlivými bity. Byly vyvinuty 60. letech 20. století. Proces dekódování, který je výpočetně nejnáročnější, se může rozdělit do několika fází. Každou lze realizovat několika algoritmy. Jelikož algoritmy používají složité matematické operace, realizují se tyto kódy častěji implementací do obvodů PLD, než jako programová součást v počítači. Na trhu jsou k dostání mikroprocesory, s vnitřní strukturou zapojenou přímo pro RS kódy, čímž se dosahuje vysokých propustností obvodů (v řádech 100Mbit/s). V praxi se dnes užívají RS kódy v mnoha oblastech. Nalézt je můžeme v pevných discích, v zálohovacích médiích (CD, DVD), přenosových protokolech digitální televize (DVB-T, DVB-S), vysokorychlostních modemech (ADSL, VDSL), i v satelitních přenosech.

2.1 RS kód

RS kódy zadáváme pomocí čísel (n,k) , které popisují dané parametry kódu a zároveň určují jeho vlastnosti. Parametr k určuje kolik symbolů vstupuje do kodéru, a parametr n udává počet symbolů vystupujících z kodéru. Každý symbol se skládá z m bitů. Počet symbolů, které používáme pro zabezpečení, je $2t$ (drobněji v [1]). Proměnná t udává kolik chybných symbolů jsme schopni opravit. Z toho můžeme vyvodit příslušné parametry kódu. Na obr. č. 1 je znázorněna zpráva, zakódovaná obecným RS kódem.

Informační poměr

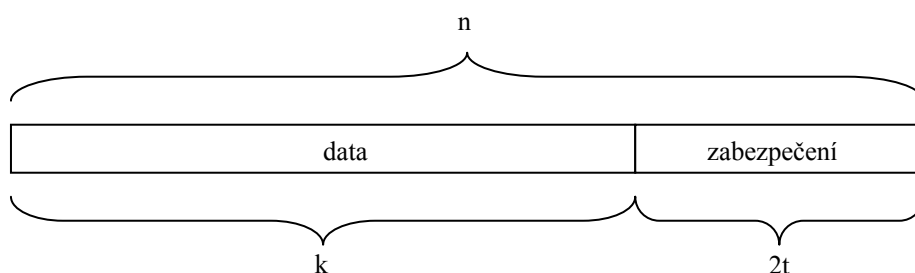
$$R = \frac{n}{k} \quad (2.1)$$

Počet symbolů pro zabezpečení

$$n - k = 2t \quad (2.2)$$

Počet symbolů, které jsme schopni opravit

$$t = \frac{n - k}{2} \quad (2.3)$$



Obr. č. 1: Struktura zprávy RS kódu

Před samotným začátkem kódování je nejprve nutné vytvořit vytvářecí mnohočlen $g(x)$, který se používá při kódování. Mnohočlen $g(x)$ se skládá z $2t$ členů.

2.1.1 Galoisovo těleso

Galoisova tělesa (pojmenována podle Évariste Galoise), někdy také nazývaná konečná tělesa, obsahují pouze konečný počet elementů. Galoisovo těleso (dále jen GF) je definováno jako $GF(x^r)$, kde x je základ číselné soustavy, kterou se přenáší data, a r udává stupeň rozšíření konečného tělesa Z_x .

Konečné těleso Z_x je tvořeno zbytky po dělení celých kladných čísel prvočíslem x . Pro prvky konečného tělesa jsou definovány operace součet a součin [1]. Operaci součtu se nazývá nonekvivalence (neboli také exklusive or).

Rozšířením konečného tělesa na GF získáme množinu vektorů, o r prvcích. Každý prvek z této množiny je prvek náležící do množiny Z_x .

$$GF(x^r) = \{[a_1, a_2, \dots, a_r]; g_i \in Z_x; i = 1, 2, \dots, r\} \quad (2.4)$$

Vektor $[a_1, a_2, \dots, a_r]$ někdy vyjadřujeme také jako mnohočlen $a(x)$

$$a(x) = a_i \cdot x^0 + a_i \cdot x^1 + a_i \cdot x^2 + \dots + a_i \cdot x^{r-1} \quad (2.5)$$

Máme-li Galoisovo těleso $GF(2^4)$ rozložíme jej na mnohočleny

$$(2^{15} + 1) = (x + 1) \cdot (x^2 + x + 1) \cdot (x^4 + x^3 + x^2 + x + 1) \cdot (x^4 + x + 1) \cdot (x^4 + x^3 + 1) \quad (2.6)$$

Protože máme $GF(2^4)$, hodí se pro nás pouze mnohočleny stupně $r = 4$. Jeden z těchto mnohočlenů, který si vybereme následně nazveme vytvářecím mnohočlenem $g(x)$. Vytvářecí mnohočlen pak vyjádříme ve tvaru

$$g(x) = (x - \alpha^0) \cdot (x - \alpha^1) \cdot \dots \cdot (x - \alpha^{2^t-1}) \quad (2.7)$$

Pro samotný proces kódování musíme vyjádřit zprávu nesoucí informaci ve tvaru

$$f(x) = f_0 + f_1x + \dots + f_{n-1}x^{n-1} \quad (2.8)$$

Každý koeficient f_{k-i} ($i = 0, 1, 2, \dots, k$) je m -bitový symbol Galoisova tělesa $GF(2^m)$. Máme-li vytvářecí mnohočlen, můžeme zakódovat zprávu [2.8]

$$\frac{f(x) \cdot x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)} \quad (2.9)$$

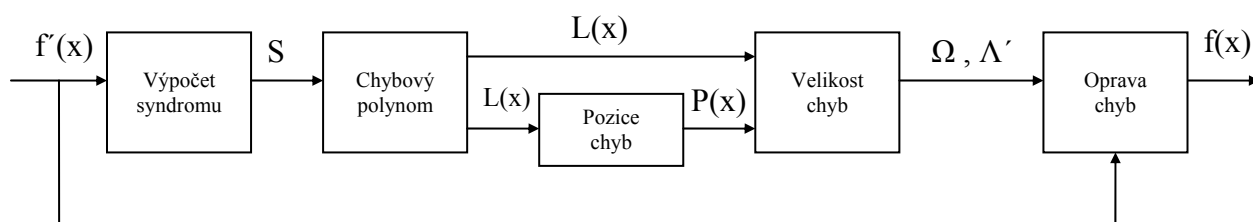
Z tohoto vzorce obdržíme $q(x)$ a $r(x)$. Hodnota $q(x)$ vyjadřuje výsledek po dělení a $r(x)$ zase zbytek.

Poté co máme zprávu zakódovanou, můžeme ji vyslat na médium jako posloupnost $f(x)$. Během přenosu můžou být přenášena data napadena chybou $e(x)$. V přijímači dekodéru následně obdržíme chybně přijatá data označená jako $f'(x)$.

$$f'(x) = f(x) + e(x) \quad (2.10)$$

2.2 RS dekodér

Dekodér RS kódu je složitější, a dá se rozdělit do několika fází. Základní rozdělení dekodéru by se dalo rozdělit na *soft* a *hard decision*. Rozdílné mezi těmito dvěma způsoby je to, že u *hard* je nejprve rozhodnuto, jestli byla přijata log 1 nebo 0, a poté se započne s dekodovacím procesem. Kdežto u *soft* je při dekodování užito přídatných informací, takže dosahujeme při dekodování lepších výsledků. Podle lit. [1] lze získat výsledek o 2dB lepší.



Obr. č. 2: Postup dekodování RS kódu

Jednotlivé bloky se dají realizovat různými algoritmy, i když provádějí stejnou operaci.

Výpočet syndromu – v případě, že vypočtené syndromy nejsou nulové, došlo k chybě a musíme provést korekci.

Chybový polynom – stanovení lokátoru chyb, kořeny lokátoru určují pozice chyb v přenášené zprávě. Pro tuto fázi se může užít více algoritmů (Berlekamp – Masey, Euklidový, ...)

Pozice chyb – nalezení kořenů chybového mnohočlenu

Velikost chyb – stanovení hodnoty chyby

Oprava chyb – oprava nalezených chyb

2.2.1 Výpočet syndromů

Jako první krok je nutné vypočítat syndromové rovnice. V případě že nenastala žádná chyba, a my jsme přijali zprávu $f(x)$, bude výsledek po dělení $f'(x)$ chybovým mnohočlenem $g(x)$ nulový. V opačném případě musíme najít rovnice S . Pro tento účel existuje způsob, kdy za koeficienty x přijaté rovnice $f'(x)$ dosadíme koeficienty a_i z Galoisova tělesa. Vše opakujeme pro $i = 1, 2, 3, \dots, 2t$ kroků.

Bezchybně přijatá zpráva

$$S_1(a_1) = S_2(a_2) = S_3(a_3) = \dots = S_{2t}(a_{2t}) = 0 \quad (2.11)$$

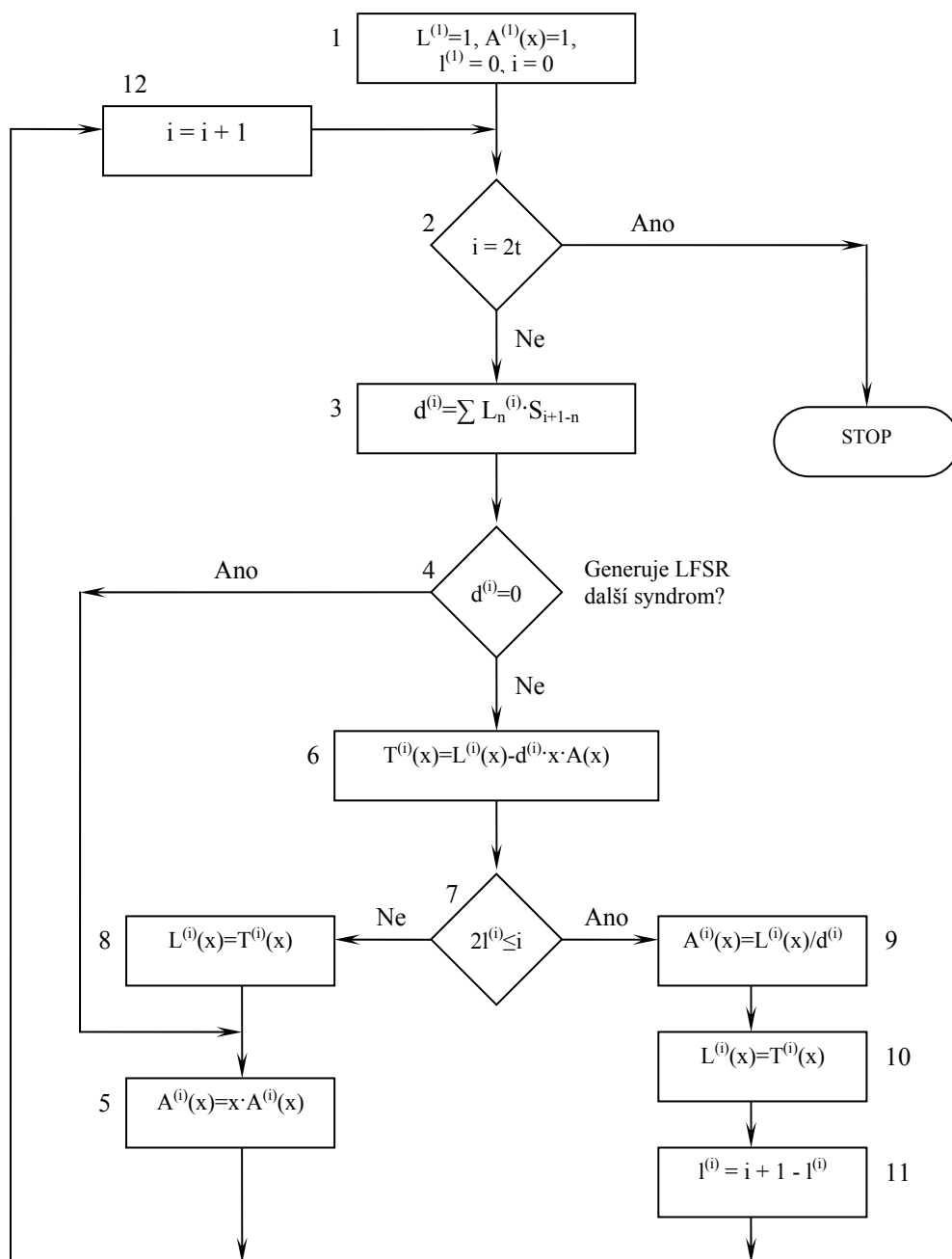
V případě že neplatí [2.11] použijeme výše popsany postup, pomocí kterého stanovíme syndromové rovnice S_i . S těmi pracujeme dále.

Pro generování syndromových rovnic jde také použít LFSR (*Linear Feedback Shifted register*). Jedná se o čítač založený na principu posuvného registru. Mimo jiné se tyto čítače používají jako generátory pseudonáhodných posloupností. Ovšem správným vnitřním zapojením získáme také generátor syndromových rovnic [2] (ne každé vnitřní zapojení LFSR můžeme použít jako generátor syndromů). Pro generování syndromových rovnic pomocí LFSR se snažíme najít takové vnitřní zapojení, které je pokud možno co nejkratší, aby generované chybové polynomy byly nejmenšího řádu.

2.2.2 Stanovení chybového polynomu - Berlekamp – Masey algoritmus

Berlekamp – Masey algoritmus (dále jen BM algoritmus), pomocí kterého nalezneme chybový polynom, je založen na postupné iteraci [1] [2]. Iterace je opakující se proces, kdy je

po stanovenou dobu vykonáván určitý výpočet, čímž dostáváme hledaný výsledek. V BM algoritmu se začíná krokem $i = 0$ a pokračujeme až do kroku, kdy $i = 2t - 1$. To je poslední krok, kterým se algoritmus ukončí. Dále na začátku nastavíme počáteční délku LFSR $l^{(1)} = 0$, chybový polynom $L^{(1)}(x) = 1$ a pomocný chybový polynom $A^{(1)}(x) = 1$. Obrázek č. 3 znázorňuje vývojový diagram BM algoritmu. Jednotlivé kroky jsou očíslovány, a jsou popsány níže [2].



Obr. č. 3: Vývojový diagram Berlekamp – Maseyho algoritmu

1. krok - nastavení počátečních proměnných
2. krok - kontrola jestli už bylo provedeno 2t iterací
3. krok - výpočet odchylky (nebo také odhad chyby), náležící k následujícímu vygenerovanému syndromu.

$$d^{(i)} = \sum_{n=0}^{l^{(i)}} L_n^{(i)} \cdot S_{i+1-n} \quad (2.12)$$

4. krok - kontrola jestli je odchylka $d^{(i)} = 0$, v případě že je rovna nule, pak současná struktura LFSR délky $l^{(i)}$ a chybový polynom $L^{(i)}(x)$ produkuje další syndrom S_{i+1} , proto také pokračujeme krokem 5. V opačném případě pokračujeme krokem 6.
5. krok - rotace pomocného LFSR. Rotujeme o tolik kroků, dokud $d^{(i)} = 0$. Nový LFSR musí být schopen generovat další syndromy a zároveň generovat již známé syndromy (vygenerované v předchozích krocích).

$$A^{(i)}(x) = x \cdot A^{(i)}(x) \quad (2.13)$$

Dále pokračujeme krokem 12.

6. krok - v případě že $d^{(i)} \neq 0$, upravíme dočasný chybový polynom $T^{(i)}(x)$ tak, že $A^{(i)}(x)$ zvětšíme a normalizujeme a odečteme od $L^{(i)}(x)$

$$T^{(i)}(x) = L^{(i)}(x) - d^{(i)} \cdot x \cdot A^{(i)}(x) \quad (2.14)$$

7. krok - kontrola jestli LFSR je třeba zvětšit, nebo ne. V případě že ne, pokračujeme krokem 8, jinak krokem 9.
8. krok - aktualizace chybového polynomu podle dočasného polynomu $T^{(i)}(x)$

$$L^{(i)}(x) = T^{(i)}(x) \quad (2.15)$$

Pokračujeme krokem 5.

9. krok - normalizujeme poslední chybový polynom $L(x)$ dělením $d^{(i)} \neq 0$ a uložíme výsledek do pomocného LFSR $A^{(i)}(x)$

$$A^{(i)}(x) = \frac{L^{(i)}(x)}{d^{(i)}} \quad (2.16)$$

Pokračujeme krokem 10.

10. krok – nyní můžeme přepsat $L^{(i)}(x)$. poté co byl normalizován a uložen $A^{(i)}(x)$ během kroku 9, proto obnovíme polynom $L^{(i)}(x)$

$$L^{(i)}(x) = T^{(i)}(x) \quad (2.17)$$

Pokračujeme krokem 11.

11. krok - podle kroku 7 musí být LFSR prodloužen, to provedeme pomocí rovnice 2.18 a pomocí teoremu.

$$l^{(i+1)} = i + 1 + l^{(i)} \quad (2.18)$$

pokračujeme na krok 12

12. přičteme iterační krok i a pokračujeme na krok 2

Teorém:

LFSR délky $l^{(i)}$, který používáme pro generování syndromů S_1, S_2, \dots, S_{i-1} není potřeba zvětšovat. Proto nám postačuje použít vzorec 2.19. V případě že ale LFSR není schopen generovat potřebné syndromy, je ho třeba zvětšit podle vzorce 2.20 [2].

$$l^{(i+1)} = l^{(i)} \quad (2.19)$$

$$l^{(i+1)} = \text{MAX}\{l^{(i)}, (i + 1 - l^{(i)})\} \quad (2.20)$$

Z toho vyplývá, že LFSR by měl být zvětšen pouze tehdy, když platí následující rovnice.

$$i + 1 - l^{(i)} > l^{(i)} \quad (2.21)$$

$$i + 1 > 2l^{(i)} \quad (2.22)$$

$$i \geq 2l^{(i)} \quad (2.23)$$

V některé literatuře se uvádí jako součást Berlekamp – Maseyho algoritmu také následující algoritmus, což je Chienovo vyhledávání. Jde pouze o spojení těchto dvou algoritmů za sebe. Na jejich funkci to nemá žádný význam. Pro přehlednost jsou zde ale rozděleny do dvou samostatných bloků.

Kromě BM algoritmu můžeme také použít jiné algoritmy, jako například Euklidův algoritmus, Peterson-Gorenstein-Zierlerův algoritmus, nebo přímá metoda.

2.2.3 Pozice chyb – Chienovo vyhledávání

Pomocí Chienova vyhledávání můžeme z koeficientů chybového polynomu $L(x)$ určit pozici chyb. Metoda je založena na principu pokus a omyl. To znamená, hledáme kořeny polynomu $L(x)$, a to tak že postupně dosazujeme do polynomu $L(x)$ a testujeme zda je $L(x) = 0$. Tento postup neustále opakujeme, dokud nedosadíme všechny možné kořeny do rovnice a tím nezískáme pozice chyb $P(x)$.

2.2.4 Velikost chyb – Forneyův algoritmus

Forneyův algoritmus se používá pro stanovení velikosti chyb. Ke stanovení velikosti chyb je zapotřebí znát pozice chyb $P(x)$ z Chienova vyhledávání, a také chybový polynom $L(x)$. Každému P vypočteme velikost chyby $M(x)$. Jako první krok je výpočet rozhodovacího polynomu $E(x)$ pomocí vzorce 2.24.

$$E(x) = S(x) \cdot L(x) \quad (2.24)$$

Poté provedeme derivaci chybového polynomu $L(x)$. Všechny hodnoty pak dosadíme do vzorce 2.25, pomocí kterého určíme velikosti chyb $M(x)$.

$$M_j = \frac{E(P_j^{-1})}{P_j^{-1} L'(P_j^{-1})} \quad (2.25)$$

2.2.5 Oprava chyb

V závěrečném kroku jsou už známy potřebné informace k opravení chyb. Proto jsme schopni sestavit slovo $e(x)$, které přičteme k $f'(x)$. Tím získáme původní $f(x)$.

3. Prokládání

Prokládání (angl. interleaving) je zabezpečením proti chybám, ovšem ne v pravém slova smyslu. Používá se v součinnosti s nějakým protichybovým kódem. Princip spočívá v tom, že si načteme do paměti několik zabezpečených zpráv, které chceme vysílat. Ty rozdělíme na menší části, které vysíláme na vedení zpréházeně. Na straně přijímače se musí před dekódováním jednotlivých zpráv opět provést proces prokládání, ovšem inverzní oproti tomu ve vysílači.

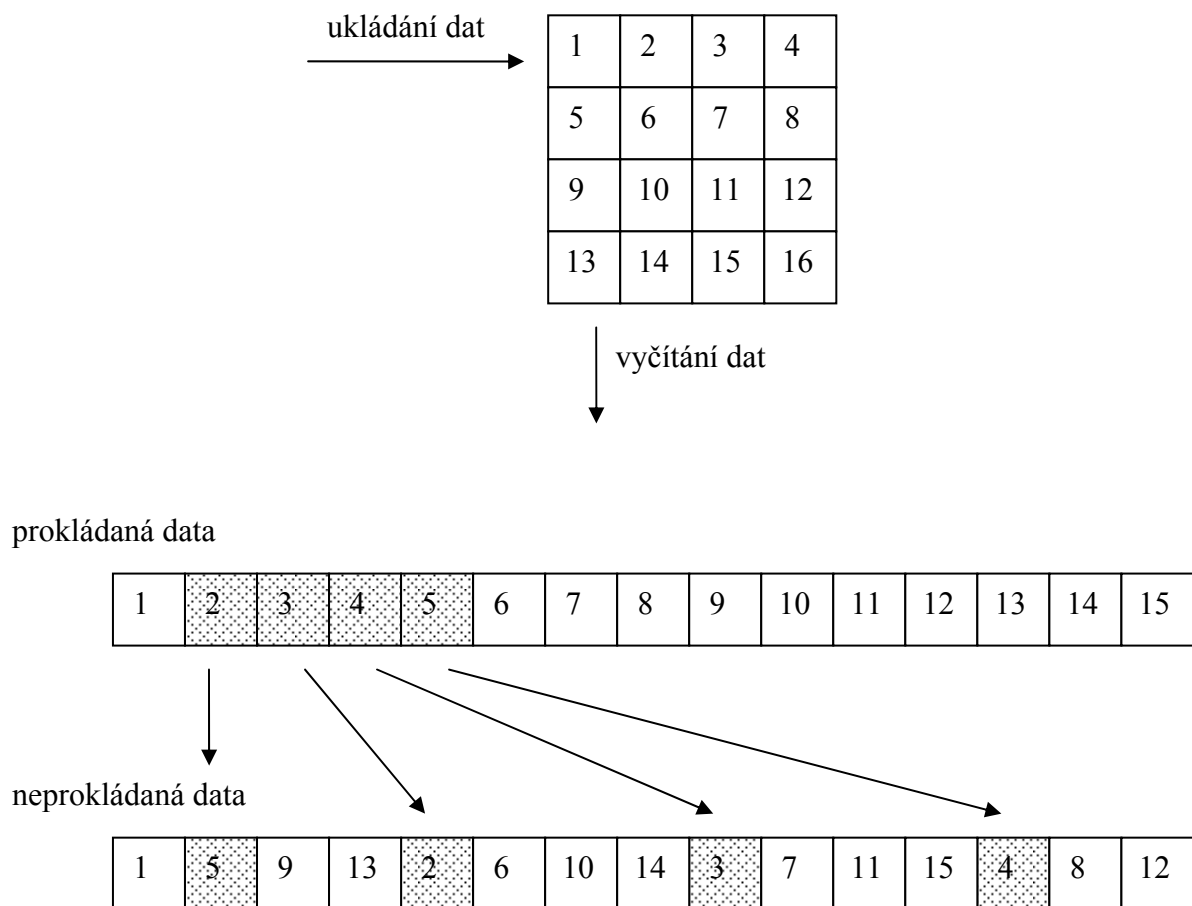
Tímto docílíme toho, že v případě výskytu dlouhého shluku chyb (nebo i krátkého výpadku přenosu), nedojde ke ztrátě např. dvou zpráv, ale k porušení pár bitů v každé zabezpečené zprávě. Počtu zpráv které před vysíláním skládáme mezi sebe se říká hloubka prokládání. Tento údaj nám také vyjadřuje kolikrát větší může být chyba, kterou jsme schopni opravit zabezpečovacím kódem, oproti případu bez použití prokládání.

V opozici této výhody, kdy jsme schopni opravit více chyb, má prokládání dvě nevýhody. První je doba zpoždění, kterou systém vytváří, a druhá je náročnost na paměť. Dnes již ceny pamětí nejsou velké, takže druhý problém není tak závažný. Ovšem zpoždění vlivem prokládání může v některých systémech (např. při multimediálním přenosu) způsobovat nemalé problémy.

3.1 Blokové prokládání

Při blokovém prokládání se nejprve čeká na naplnění paměti zabezpečenými daty, a teprve pak se systematicky vyčítá z této paměti. Na straně přijímače se pak musí opět čekat, než se přijmou všechna prokládaná data, a teprve pak se může začít s opětovným vytvořením původní zprávy. Máme-li hloubku prokládání h , tak doba zpoždění t_{zp} na straně vysílače i přijímače je dána vztahem 3.1.

$$t_{zp} = h \cdot n \quad (3.1)$$



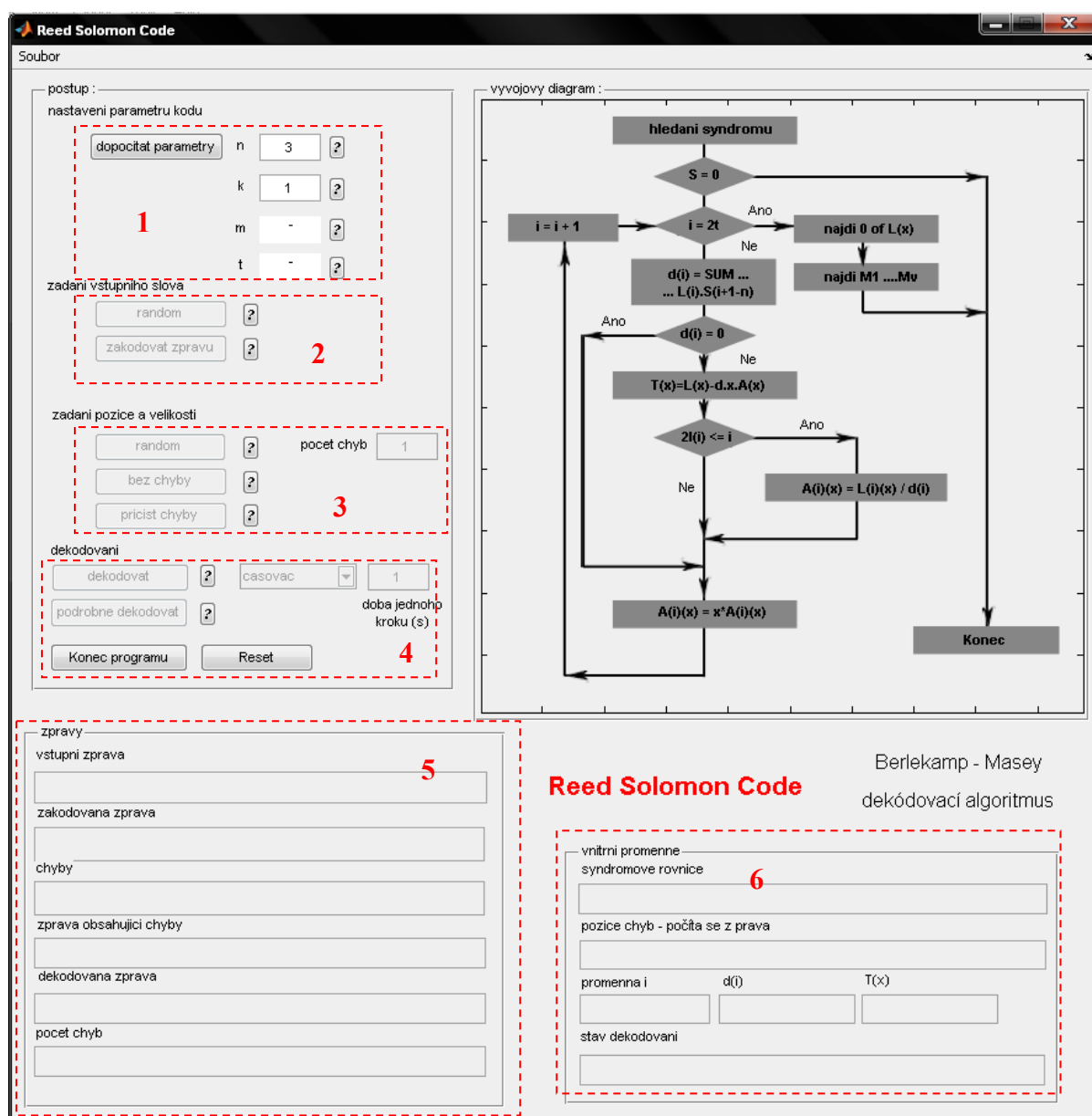
Obr. č. 4: Princip blokového prokládání, schéma rozprostření chyb

3.2 Konvoluční prokládání

Konvoluční prokládání částečně řeší problém se vzniklým časovým zpožděním. Neskládá se pouze z pamětí, ale také z různých velikých posuvných registru. Do těch se postupně zapisuje a vyčítá. Při použití konvolučního prokládání se zmenšuje doba zpoždění přibližně na polovinu oproti blokovému. Přesná doba zpoždění je dána počtem a délkou posuvných registrů a také systémem jakým do nich zapisujeme.

4. Program v prostředí Matlab

Pro znázornění dekódovacího algoritmu jsem v prostředí Matlab 7.1 vytvořil aplikaci, ve které se dá názorně vyzkoušet princip dekódování. Program je ošetřen pro případ špatného zadání, a pokud budou zadány hodnoty, které nejsou platné, objeví se informační okno s nápovědou. Nejjednodušší způsob spuštění kódování je ponechat přednastavené parametry kódu, a pouze nechat náhodně vygenerovat zprávu pro přenos a počet a velikost chyb. Při volbě podrobného dekódování se bude průběh dekódování zobrazovat ve vývojovém diagramu.



Obr. č. 5: Aplikace v prostředí Matlab

Okno aplikace lze rozdělit na několik částí. V částích 1,2,3 a 4 postupně zadáváme vstupní údaje, a v částech 5 a 6 se vypisují proměnné a dodatečné informace.

1. část - zadání parametrů kódu, který budeme používat
2. část - zadání a zakódování vstupní zprávy
3. část - zadání pozice a počtu a velikosti chyb, a jejich přičtení k zakódované zprávě
4. část - nastavení parametrů dekodování, možnost zobrazení ve vývojovém diagramu (pomocí nabídky lze vybrat jestli bude proces dekodování řízen časovačem, nebo budeme jednotlivé kroky posouvat sami), tlačítko **Reset** vrátí aplikaci do stavu po spuštění a **Konec programu** ukončí aplikaci
5. část - zobrazení zakódované zprávy, zprávy s chybou a dekodované zprávy, popřípadě lze také ručně zadat zprávu nebo chyby
6. část - zobrazení pomocných proměnných, které se používají při dekodování, a popis stavů, ve kterých se dekodovací proces nachází

V případě že by uživatel neznal význam jednotlivých proměnných, může pomocí tlačítka se symbolem ? dozvědět jejich funkci a význam.

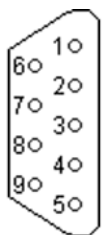
5. Zabezpečení komunikace přes RS232

I přesto, že rozhraní RS232 označované také jako sériová linka, bylo standardizováno na konci 70. let minulého století, dodnes se běžně využívá. Ve velké míře se s ním můžeme setkat například při konfiguraci různých přístrojů (výrobní linky, informační panely, televize,...), připojení k modemům a nebo při vzájemné komunikaci mikroprocesorů.

5.1 Popis rozhraní RS232

Rozhraní rs232C, což byla jeho poslední úprava, bylo standardizováno v roce 1969. Vyskytuje se téměř na všech stolních počítačích a na některých přenosných. Pokud není k dispozici, je k sehnání redukce přes rozhraní USB.

Pro rozhraní je definováno několik typů konektorů, a to CANNON 9, CANNON 25 a RJ45. Dnes se nejčastěji setkáváme s CANNON 9, který je na obr. č. 6. K dispozici je devět pinů, které mají specifické funkce. Význam jednotlivých pinů je v tab. č. 1.

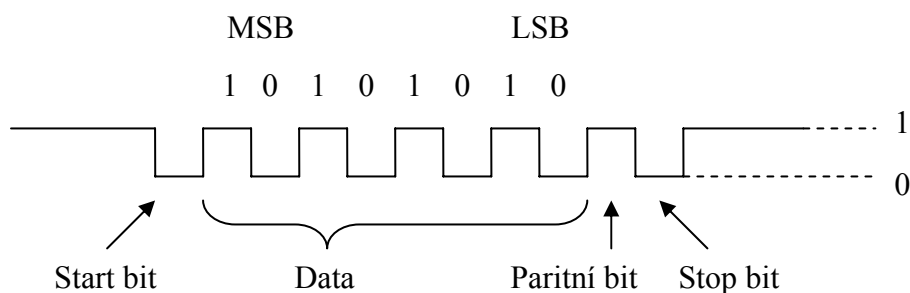


Obr. č. 6: Konektor CANNON 9 rozhraní RS232

Tab. č. 1: Funkce jednotlivých pinů rozhraní RS232

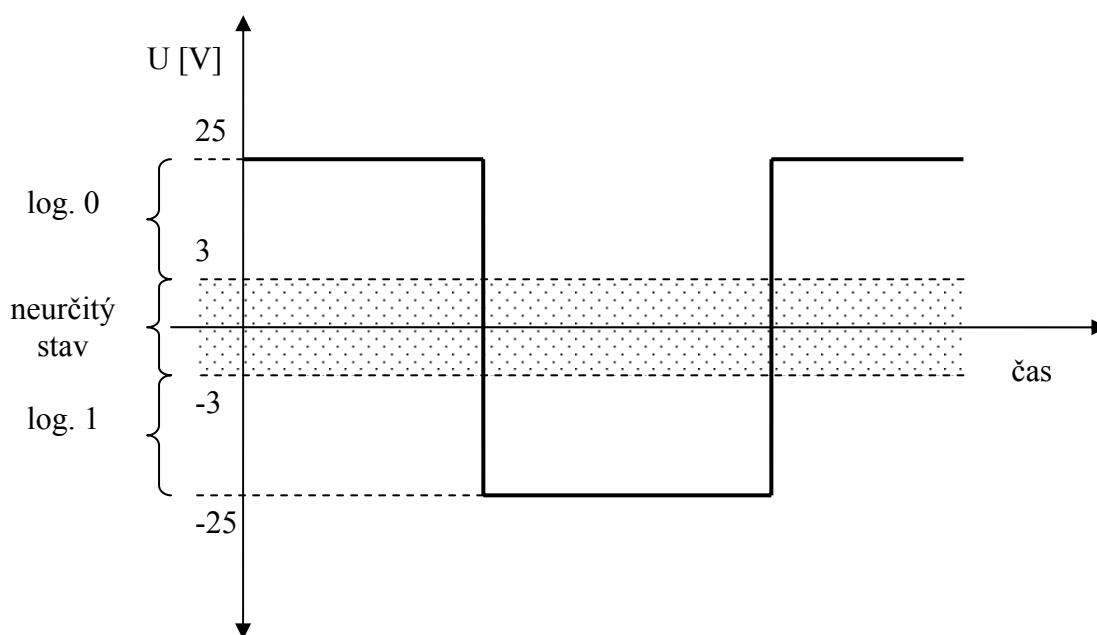
pin	název	směr dat	význam
1	CD	<--	příjem nosného signálu
2	RXD	<--	příjem dat
3	TXD	-->	vysílání dat
4	DTR	-->	signál že zařízení je připraveno
5	GND	-	zem
6	DSR	<--	komunikační kanál je volný
7	RTS	-->	oznámení o požadavku vysílat
8	CTS	<--	vysílání nosného signálu
9	RI	<--	detekce volání

Pro samotný přenos dat slouží pouze dva piny , a to jsou RXD a TXD. GND značí zem, nebo také nulu. Zbylé signály slouží pro řízení komunikace mezi dvěma zařízeními, např. modemem a počítačem. Přenos dat přes RXD a TXD probíhá asynchronně, což znamená že před vysíláním daty je nejprve úvodní znak (start bit), pomocí jehož sestupné hrany se synchronizuje následné časování. Pak jsou přenášena data (7 nebo 8 bitů) a ta jsou ukončena ukončovacím znakem (stop bit). Pro některá pomalejší zařízení se používá zdvojení stopbitu na dva. Zařízení, které přijatá data zpracovává tím má dostatečnou dobu, aby bylo schopno přijímat další data. Dále se ještě může použít kontrola správnosti přenosu pomocí paritního bitu. Je to nejjednodušší způsob ověření správnosti. Můžeme zvolit sudou nebo lichou paritu. Zvolíme-li například sudou paritu, nastaví se paritní bit tak, aby součet všech jedničkových bitů plus paritní bit dal sudé číslo. Všechny tyto parametry a ještě rychlost přenosu se nastavují při vytváření spojení. Struktura přenášených dat je zobrazena na obr. č. 7.



Obr. č. 7: Struktura přenášených dat

Je-li vyslána na vedení log. 0, je přenášena jako kladná hodnota napětí, naopak log. 1 je reprezentována záporným napětím. Pro log. 0 vysílač vysílá impulsy v intervalu napětí od +5V do +15V, přijímač detekuje toto napětí v intervalu +3V do +25V. Pro log. 1 platí stejné intervaly, ale se zápornými hodnotami napětí. V rozmezí od +3V do -3V je pásmo neurčitého stavu.



Obr. č. 8: Napět'ové úrovně detekce logických úrovní rs232

Propojíme-li piny RXD a TXD, dostaneme zapojení zpětné smyčky (loopback), čímž docílíme přeposílání odesílaných dat přímo na vstup. Tím jednoduše můžeme ověřit správnou funkci rozhraní, např. v programu hyperterminal. Rychlost přenosu se nastavuje v Bd/s. Tedy počet možných stavů, přenesených za sekundu. Do této rychlosti jsou započítány start-bity, stop-bity paritní i datové bity. Rychlost, jakou přenášíme datové bity je tedy dána vztahem 5.1.

$$rychlost [bit / s] = \frac{8 \cdot \text{datový bit}}{1 \cdot \text{start bit} + 8 \cdot \text{datový bit} + 1 \cdot \text{stop bit} + 1 \cdot \text{paritní bit}} \cdot \text{nastavená rychlost} [Bd / s] \quad (5.1)$$

Výsledná rychlost se může zvyšovat nebo snižovat, závislosti na celkovém počtu datových bitů v 1 slově, tedy kolik stopbitů je použito a zda je použita parita. V tab. č. 2 jsou uvedeny některé případy přenosových rychlostí, a hodnoty registru, které nastavují přenosovou rychlost portu. Při praktickém implementaci kódu, popsané v kapitole 5.4 a 5.5 jsem použil rychlost 115 200 Bd/s. Informace se tedy podle vzorce 5.1 po lince přenášejí rychlostí 11,520 kB/s. Jde o přenos dat i s režijními daty (data + zabezpečení + synchronizace + hlavička).

Tab. č. 2: Přenosové rychlosti

rychlost [bd/s]	konstanta	3F9	3F8
110	1024	04	17
150	768	03	00
300	384	01	80
600	192	00	c0
1200	96	00	60
2400	48	00	30
4800	24	00	18
9600	12	00	0c
19200	6	00	6
115200	0	00	00

5.2 Použitý zabezpečovací kód

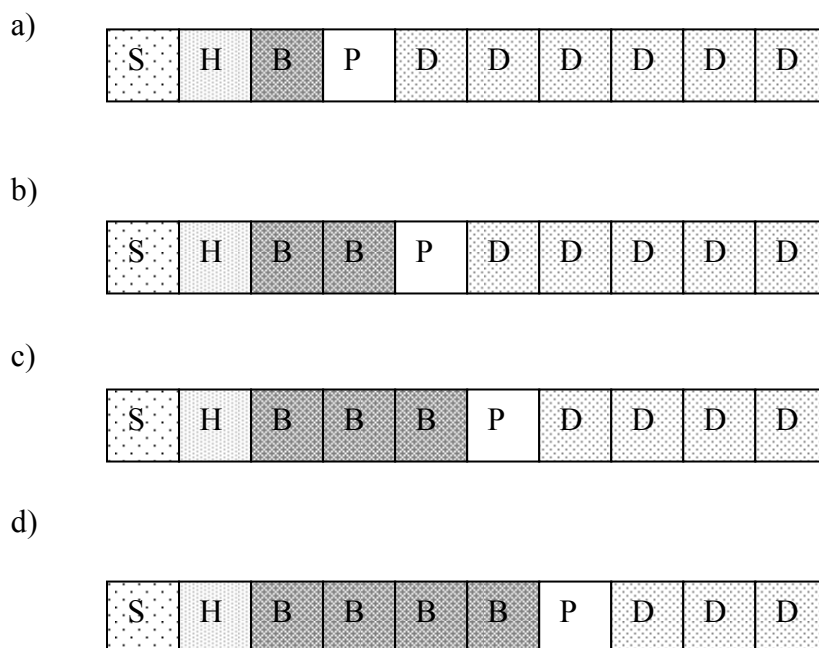
Pro zabezpečení jsem zvolil Reed Solomonův kód, a to ve čtyřech variantách, navíc také s možností prokládání. Parametry kódu jsou uvedeny v tab. č. 3. Maximálně je v jednom zabezpečeném bloku přeneseno datových 13 znaků, naopak při užití maximálního zabezpečení lze přenést pouze 7. V případě použití prokládání jsou data prokládána s hloubkou 8krát.

Tab. č. 3: Použité kódy a jejich vlastnosti

typ kódu	počet datových bitů	počet zabezpečovacích bitů	počet opravitelných chyb
15 / 13	13	2	1
15 / 11	11	4	2
15 / 9	9	6	3
15 / 7	7	8	4

5.3 Struktura použitého protokolu

Navržený protokol se přenáší po rámcích, které se skládají z 10ti bajtů. Každý bajt je přenášen přes sériovou linku takovým způsobem, že je na začátku jeden start bit, pak následuje 8 datových bitů, paritní bit není použit, a na závěr je zařazen jeden stop bit. K přenosu jednoho bajtu je tedy zapotřebí 10ti bitů. Prvním bajtem rámce je přenášena synchronizace, druhým záhlaví, a zbylými bajty jsou přenášena zabezpečená data. Uvedená struktura rámce je zobrazena na obr. č. 9, a platí pouze v případě neprokládaného přenosu. Případ, kdy se data přenášejí prokládaně, bude popsán v kapitole 5.3.1.



Obr. č. 9: Struktura přenášeného rámce - a) kód RS 15/13 b) kód RS 15/11 c) kód RS 15/9 d) kód RS 15/7 ; S - synchronizace , H - záhlaví , B - zabezpečení , P – pomocná data, D - data

Synchronizace – vyskytuje se jako první bajt přenášeného rámce, má strukturu 01010101B, tedy v dekadickém vyjádření je to 85 a v ASCII kódu tento znak reprezentuje „U“.

Záhlaví – je přenášeno druhým bajtem rámce. Nese v sobě informaci jaký kód byl použit, zda-li bylo použito prokládání, a o jaký typ zprávy se jedná. V případě že bylo použito prokládání, je přenášena také informace, zda jsou všechny bajty pro data využity, nebo jsou-li vyplněny výplňovými znaky. Typ zprávy definuje zda jde o přenášená data, potvrzení o bezchybném doručení, nebo o žádost o opětovné vyslání. Přesná struktura záhlaví je na obr. č. 13, tab. č. 4 zobrazuje významy jednotlivých bitů.

MSB						LSB	
1	0	0	0	0	1	0	1
typ zprávy		délka prokládání				typ kódu	

Obr. č. 10: Struktura záhlaví

Tab. č. 4: Význam jednotlivých bitů v záhlaví

128	64	32	16	8	4	2	1	význam dat	
0	0	-	-	-	-	-	-	typ zprávy	data
0	1	-	-	-	-	-	-		potvrzení
1	0	-	-	-	-	-	-		nevyužito
1	1	-	-	-	-	-	-		opakovat přenos
-	-	0	0	0	1	-	-	typ prokládání	bez prokládání
-	-	0	0	0	0	-	-		nevyužito
-	-	0	0	1	0	-	-		přenos 8 bajtů
-	-	0	1	0	0	-	-		přenos 16 bajtů
-	-	0	1	1	0	-	-		přenos 24 bajtů
-	-	1	0	0	0	-	-		přenos 32 bajtů
-	-	1	0	1	0	-	-		přenos 40 bajtů
-	-	1	1	0	0	-	-		přenos 48 bajtů
-	-	1	1	1	0	-	-		přenos 56 bajtů
-	-	-	-	-	-	0	0	typ kódu	15 / 13
-	-	-	-	-	-	0	1		15 / 11
-	-	-	-	-	-	1	0		15 / 9
-	-	-	-	-	-	1	1		15 / 7

Podíváme-li se například na ASCII tabulku, zjistíme že každý jeden znak je reprezentován jedním bajtem (tedy osm bitů). Protože je použit kód s $m = 4$, může mít jeden znak pouze čtyři bity. Proto všechny znaky převedeme z osmi-bitových na dva čtyř-bitové a následně je zabezpečíme daným kódem. Jelikož se ale data přes sériovou linku přenášejí po osmi bitech, sloučíme data a zabezpečení zpět do osmi-bitových čísel.

Použijeme-li například kód RS 15/13, můžeme zabezpečit 13 čtyř-bitových znaků. Protože ale jeden osmi-bitový znak dělíme na dva čtyř-bitové, obsadíme jich tedy pouze 12. Při generování zabezpečení tento jeden čtyř-bitový znak necháváme nulový. Po zabezpečení obdržíme zabezpečenou zprávu o délce 15ti čtyř-bitových znaků. Převedeme-li je na osmi-bitové, opět obdržíme další redundantní čtyř-bitový znak.

Tyto dva znaky jsou z hlediska zabezpečení dat zbytečné. Přenáší se v nich ale pomocná informace, sloužící k tomu, zda-li není některý z přenášených bajtů nulový. Tomu stavu se musíme vyvarovat, protože při příjmu a následném zpracování nulového znaku dochází ke vzniku chyb. Je to dáno tím, že v ASCII tabulce hodnota 0 reprezentuje řídicí znak, a ne písmeno (nebo číslici).

Zabezpečení a data – z výše popsaného a při použití kódu 15 / 13 vyplývá, že po 2 bajtech (synchronizace a záhlaví) následuje jeden bajt se zabezpečením, pak jeden bajt pro umístění nulových bajtů a šest datových bajtů na závěr. Viz. obr. č. 9 a.

Tab. č. 5: Přenosové rychlosti dat při použití kódů z tab. č. 3

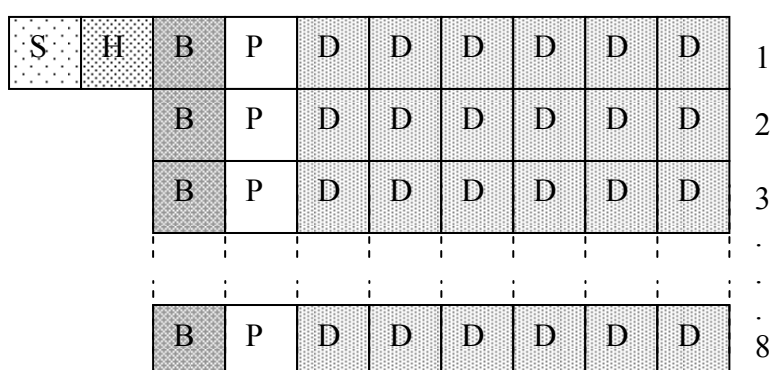
typ kódu	prokládání	délka rámce [Byte]	počet datových bitů v rámci [bit]	Přenosová rychlost dat [kB/s]
15 / 13	ne	10	6	6,912
15 / 11	ne	10	5	5,760
15 / 9	ne	10	4	4,608
15 / 7	ne	10	3	3,456
15 / 13	ano	66	48	8,378
15 / 11	ano	66	40	6,922
15 / 9	ano	66	32	5,585
15 / 7	ano	66	24	4,189

Ze vztahu 5.1 a vlastností kódu z tab. č. 3, lze vypočítat rychlost přenosu dat. Uvedená rychlost udává rychlost přenosu dat, nikoli i režijních informací (synchronizace, záhlaví).

5.3.1 Struktura použitého protokolu při prokládání

Je-li užito prokládání, změní se struktura celého přenášeného rámce. Je zbytečné, aby se opakovaně přenášely bajty záhlaví a synchronizace. Proto jsou posílány pouze jednou na začátku přenosu. Pak už jsou prokládaně přenášeny bajty zabezpečení a dat. Použil jsme prokládání s hloubkou osm. Struktura rámce je na obr. č. 11.

V případě že máme méně dat k odeslání, než je možné přenést jedním prokládaným rámcem, obsadí se zbývající nevyužitá místa výplňovými bajty. Abychom jsme je po dekódování dále nepředávali jako data, je informace, o tom kolik bajtů je využito a kolik ne, přenášena v pomocných datech. Nevyužitá bajty musíme bohužel přenášet, abychom splnili požadavek prokládání.

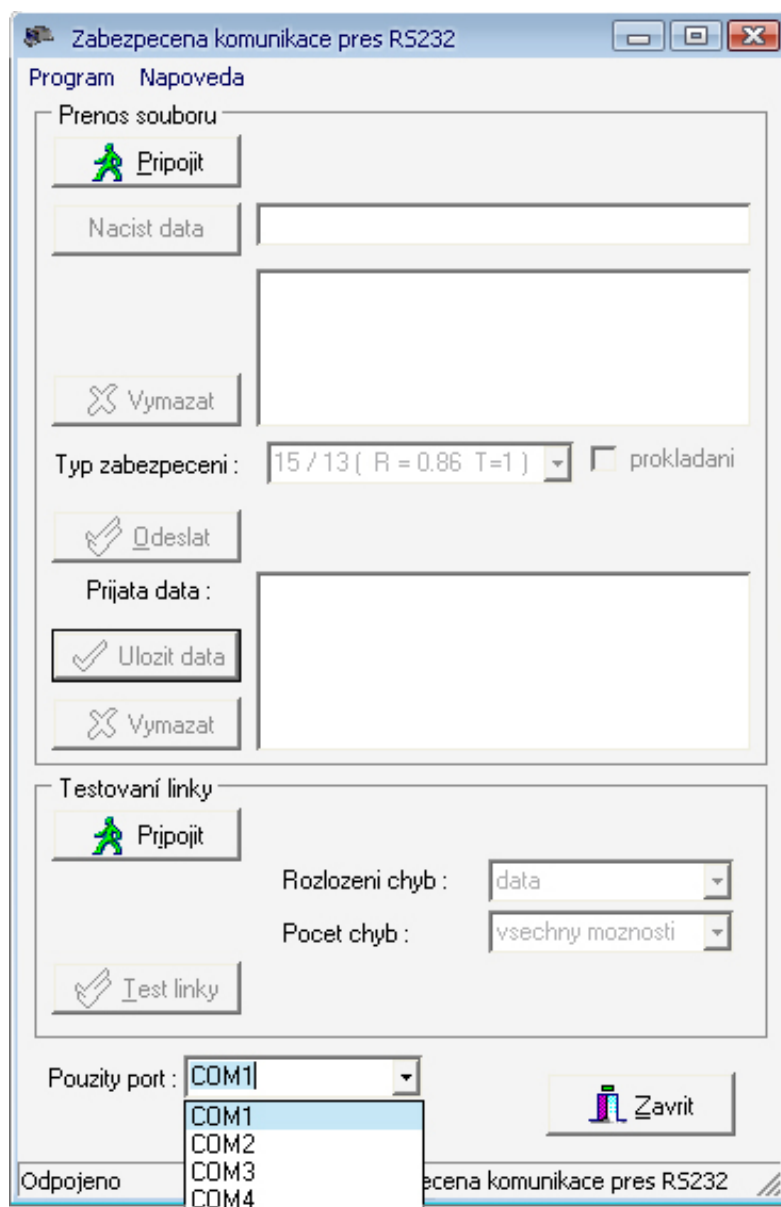


Obr. č. 11: Struktura prokládaného rámce při použití kódu RS 15/13

Prokládaný rámec má délku 66 bajtů, kdežto neprokládaný má pouze 10. Ale pro přenesení stejného množství dat v neprokládaném režimu, potřebujeme 80 bajtů. Rozdíl 14ti bajtů je úspora zmenšením množství režijních dat (synchronizace a záhlaví).

5.4 Program pro PC

Program jsem tvořil v prostředí Borland Builder 6.0 , a psal jsem jej v jazyce C++. Je spustitelný na každém počítači s operačním systémem Windows. Při programování jsem si doinstaloval do programovacího prostředí komponentu TCommPort¹⁾ , pomocí ní je umožněn přístup k sériové lince na pc. Pro obsluhu jsou nutné alespoň minimální znalosti pc a práce s jeho rozhraními.



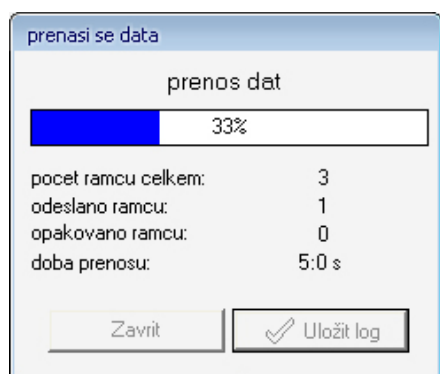
Obr. č. 12: Hlavní okno programu rs.exe , výběr COM portu

¹⁾ výrobce: xtrimsoft , verze 1.0 , informace: <http://www.torry.net/authorsmore.php?id=4508> , stáhnutí v zip souboru: <http://www.torry.net/vcl/comms/modems/CommPortC5C6.zip>

Program není třeba instalovat, skládá se z jednoho spustitelného souboru *rs.exe* . Po jeho spuštění se zobrazí hlavní okno programu (viz. obr. č. 12). Ve spodní části okna zvolíme, ke kterému com portu máme připojen mikropočítač (máme na výběr COM1 až COM4). V případě že nyní chceme přenášet data, zmáčkneme tlačítko **Připojit** v horní části okna. Chceme-li naopak nejprve otestovat chybovost linky, zmáčkneme **Připojit** ve spodní části. Program je kdykoli možné ukončit stisknutím tlačítka **Zavřít**.

Přenos souborů : po připojení se zpřístupní tlačítka k přenosu souborů. Nejprve klikneme na tlačítko **Nacist data** , a vybereme textový soubor, který chceme přenášet (jiné než textové soubory nelze přenášet). Po potvrzení volby se zobrazí cesta k tomuto souboru a níže se pak vypíše jeho obsah. Maximální velikost souboru, který chceme přenášet je 1MB. V případě že jsme zvolili špatný soubor, stiskneme vedlejší tlačítko vymazat, které vymaže načtený soubor. Následně opakujeme výběr souboru pro odeslání. V dalším kroku zvolíme jeden ze čtyř typů kódu, kterým chceme zabezpečovat data, a zatrhneme možnost **prokládání**, jestliže jej chceme použít.

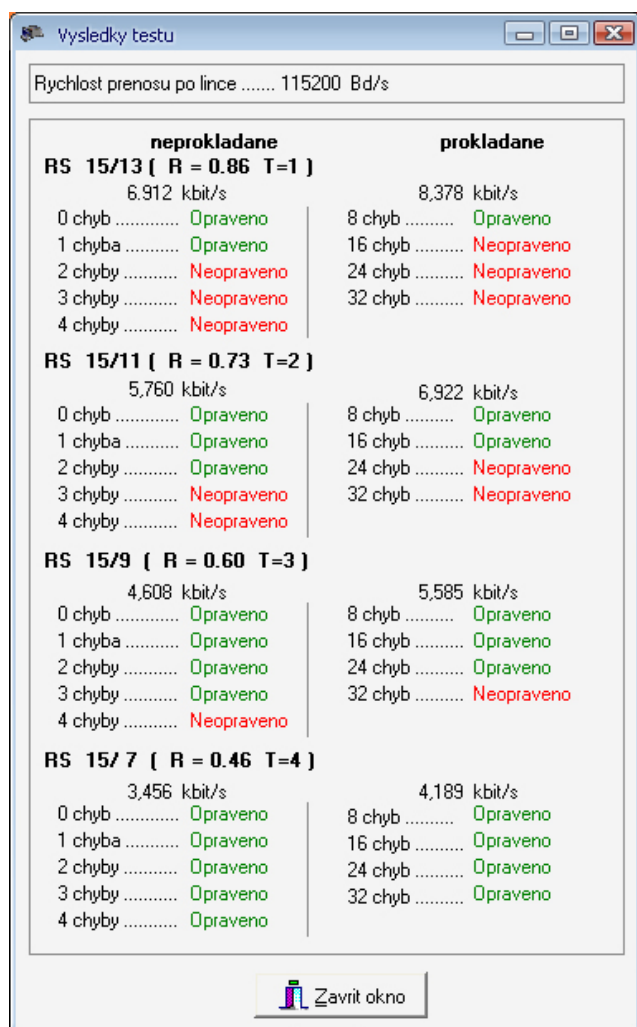
Během přenosu se zobrazí okno (viz. obr. č. 13) , ve kterém je zobrazen samotný průběh přenosu. V horní části se zobrazuje kolik procent dat už bylo přeneseno. Níže jsou pak zobrazeny počty rámců, a to kolik je jich celkem, kolik rámců už bylo odesláno, a kolik z odeslaných rámců se muselo opakovat. Po ukončení přenosu se zpřístupní tlačítko **Zavřít** (skryje okno), a **Uložit log** (uloží výsledky přenosu souboru to textového souboru log.txt).



Obr. č. 13: Okno zobrazující průběh přenosu dat

Zavřením okna s výsledky přenosu se opět dostáváme do stavu, kdy můžeme načíst jiná data a opětovně je odeslat. V případě příjmu dat jsou dekodovaná data zobrazena v prostřední části hlavního okna programu. Můžeme je smazat tlačítkem **Vymazat** a nebo uložit tlačítkem **Uložit data** (zobrazí se dialogové okno pro uložení souboru).

Testování linky : po připojení se zpřístupní tlačítka k testování linky. Ta jsou ve spodní části hlavního okna programu. Zvolíme rozložení a počet chyb. Test spustíme tlačítkem **Test linky**. Po dokončení se zobrazí okno s výsledky (viz Obr. č. 14). Levý sloupec zobrazuje výsledky neprokládaných kódů, pravý naopak prokládaných.

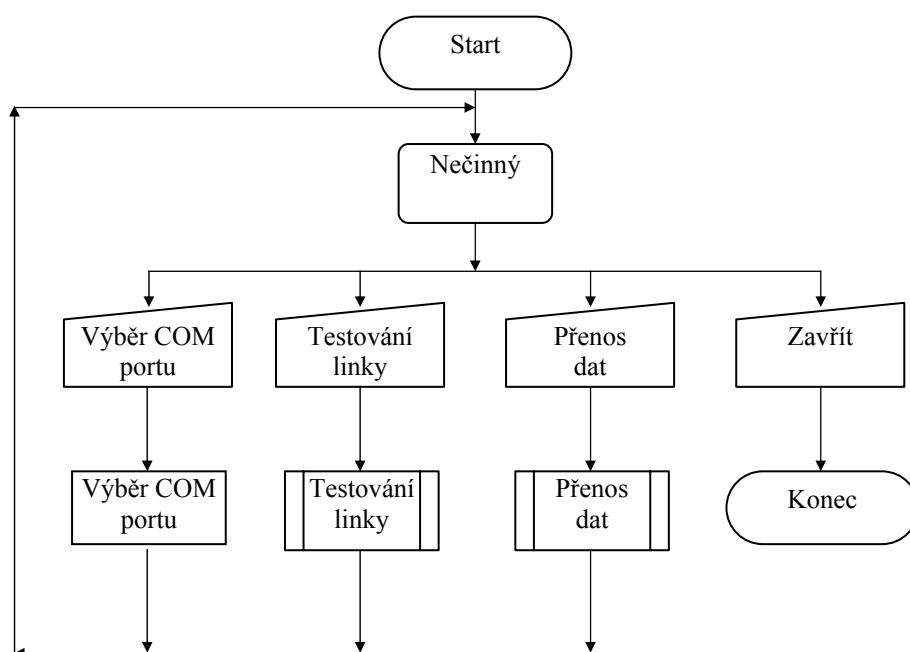


Obr. č. 14: Okno s výsledky testování chybovosti linky

Z parametrů jednotlivých kódů víme, kolik chyb by měli opravit a kolik již ne. Proto u prokládaných kódů nejsou zobrazeny všechny možné chyby, ale pouze důležité hodnoty. Okno s výsledky zavřeme tlačítkem **Zavřít okno**. Tím se dostaneme zpět do stavu před spuštěním testu na chybovost linky.

5.4.1 Samotná funkce programu

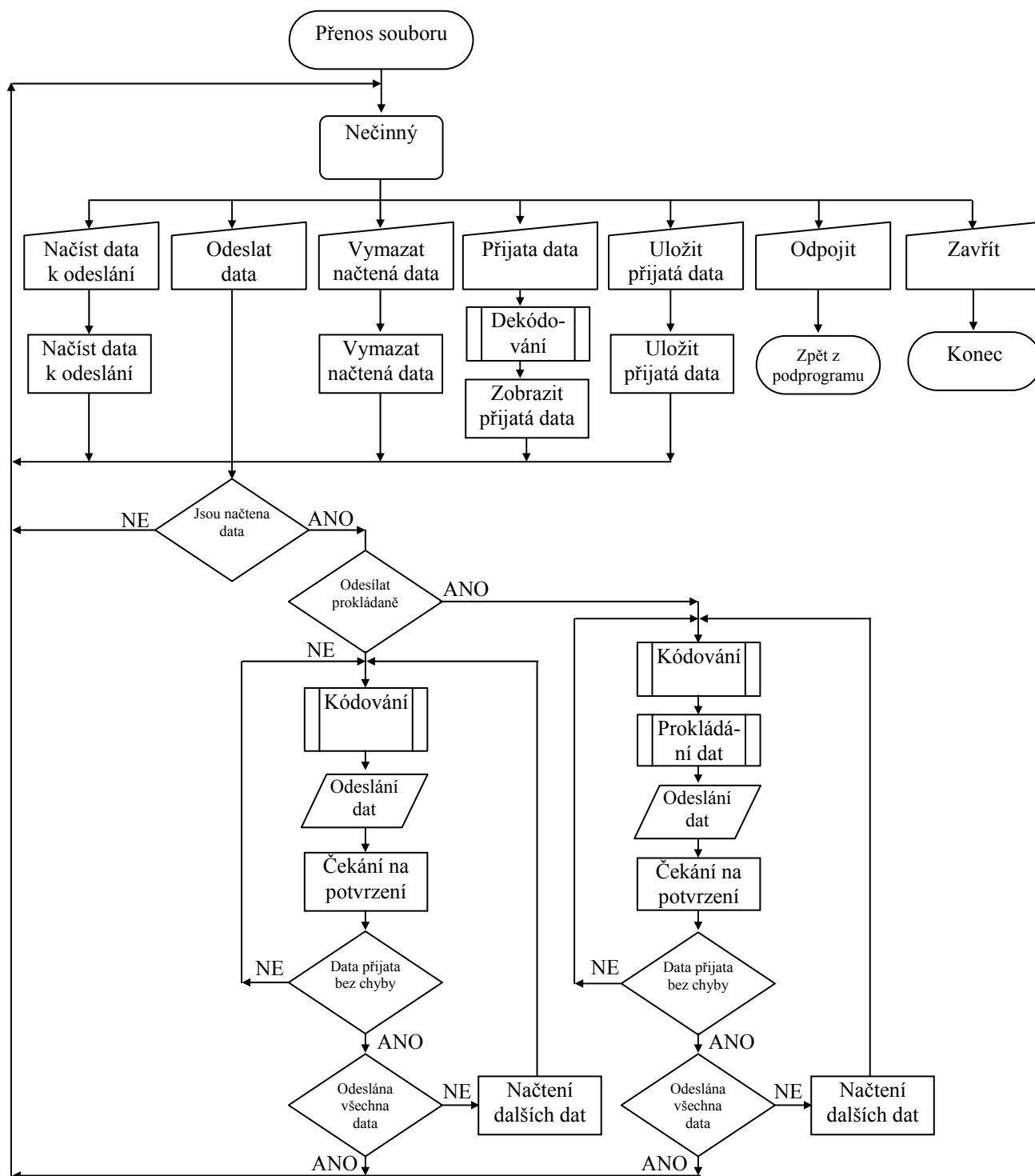
Vzhledem k tomu, že se jedná o grafické rozhraní, byl by úplný a kompletní vývojový diagram velice rozsáhlý. Proto vývojový diagram je rozdělen do několika částí, a některé události nejsou zobrazeny vůbec. Je to například zobrazení nápovědy, nebo informací o programu.



Obr. č. 15: Vývojový diagram – start programu

Průběh vývojového diagramu se téměř shoduje s popisem obsluhy programu uvedený v 5.4, ale vývojový diagram zobrazuje více informací o jednotlivých fázích kódování a odesílání. Program po startu přejde do stavu nečinný, a čeká na událost, kterou vyvolá uživatel. Kromě ukončení programu je možné nastavit port, kterým budeme komunikovat, přejít do podprogramu přenosu souborů, nebo testování chybovosti linky. Vývojový diagram Testování linky a přenosu souborů je zobrazen na obr. č. 16 a obr. č. 17. Ukončení programu je možné uskutečnit téměř v jakémkoli stavu programu, i ve stavu, když je program připojený na sériovou linku. V takovém případě je automaticky nejprve provedeno odpojení, a až následně se program ukončí.

V podprogramu přenosu souborů se opět čeká na událost, kterou vyvolá uživatel. Je zde ještě možnost, že je událost vyvolána příchodem dat po lince, v takovém případě jsou data dekodována a zobrazena. Algoritmus dekodování a opravy chyby je vysvětlen v kapitole 2.2.

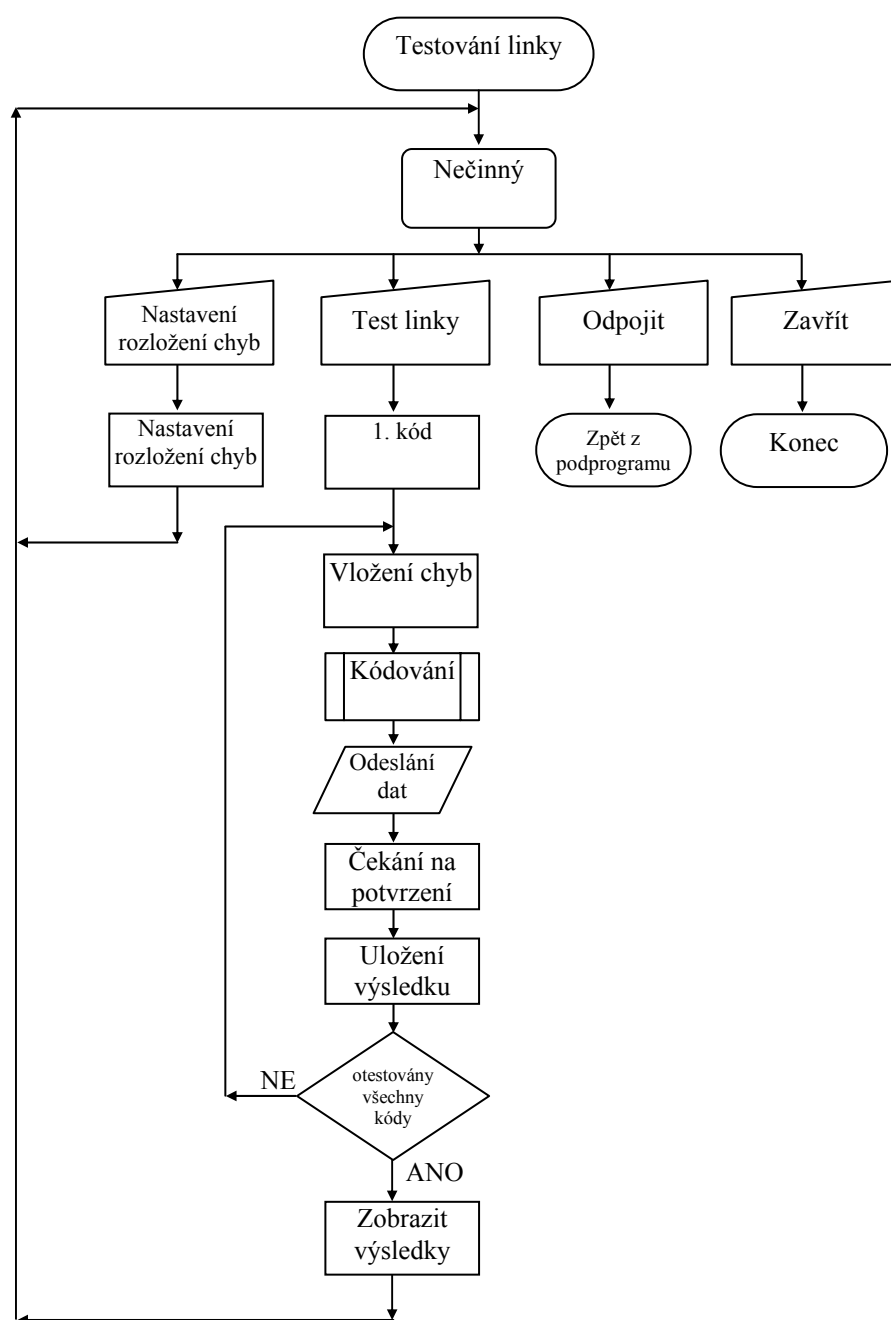


Obr. č. 16: Vývojový diagram – odesílání dat

Následně opět program čeká na příchod události. Chceme-li odeslat data, program nejprve zkontroluje jestli jsme nějaká data pro odeslání načetli. Jestliže ano, tak podle naší volby, jestli chceme data odesílat prokládaně nebo ne, přejde program do příslušné odesílací smyčky. V té jsou data zakódována, a následně odeslána. Pak program čeká na potvrzení dat. V případě kladného potvrzení byla data přenesena v pořádku a můžou se odeslat další. Při

přijmu záporného potvrzení se odešlou opět ta samá data. Informace o typu zprávy, tedy o tom zda-li byl přenos úspěšný či nikoli, je přenášena pomocí prvních dvou nejvýznamnějších bitů v záhlaví (viz. tab. č. 4). Odesílají-li se data prokládaně, funguje odesílací smyčka stejně jako u neprokládaných dat, s rozdílem že mezi blok kódování a odeslání je vložena funkce prokládání dat.

V podprogramu testování chybovosti linky se čeká na událost, kterou vyvolá uživatel. Kromě odpojení a ukončení lze nastavit počet a rozmístění chyb, a spustit test. Ten probíhá



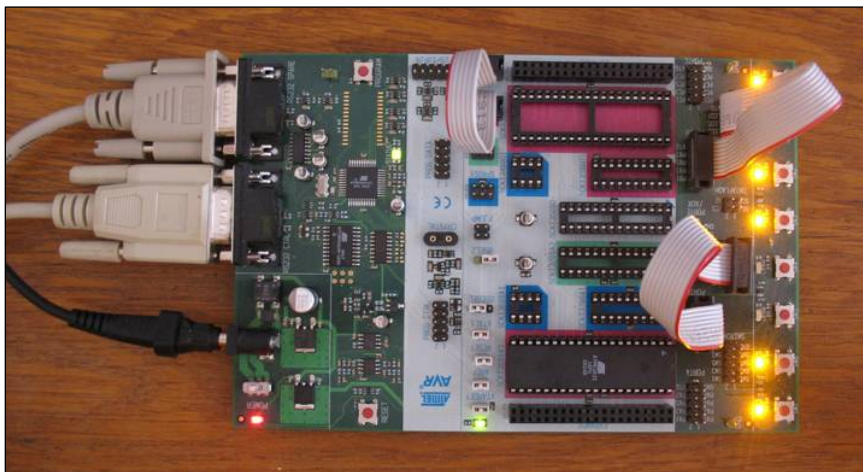
Obr. č. 17: Vývojový diagram – testování chybovosti linky

v několika krocích. Nejprve se vybere první kód a přenos bez prokládání. Následně se vypočítají zabezpečovací informace. Pak se do odesílaných dat vloží chyby, a odešlou se. V mikropočítači jsou přijatá data dekodována a vložené chyby opraveny. Výsledek o dekodování je odeslán zpět do počítače, kde se uloží. Opět se vybere stejný zabezpečovací kód, ale počet chyb, který se vkládá před odesláním dat se zvětší. Tímto způsobem otestujeme daný kód na výskyt všech chyb. V dalším kroku se zvolí druhý typ kódu a opět se provede test na všechny možné chyby. Po otestování všech kódů se zapne možnost prokládání, a opět se otestují všechny kódy na možné chyby. Vývojový diagram testování linky je na obr. č. 17. Po dokončení každého testu jsou výsledky uloženy a na závěr jsou zobrazeny (viz. obr. č. 14).

5.5 Program pro mikropočítač

Program pro mikropočítač jsem psal v jazyce C pomocí programu CodeVision AVR. Program umí provést kontrolu syntaxe, a překlad do souboru coff, pomocí něž je možné provést simulaci programu na pc. Obsahuje také rozhraní, pomocí něž lze napsaný kód naprogramovat do mikroprocesoru.

Použil jsem mikroprocesor Atmel ATmega32, osazený do desky STK 500 (obr. č.18). Jedná se o univerzální kit, pro desítky mikroprocesorů firmy Atmel. Kit obsahuje tlačítka, LED



Obr. č. 18: Kit STK 500

diody, oscilátor, rozhraní USART, A/D převodník, ISP rozhraní a spoustu dalších obvodů. Pro správnou funkci sériové linky jsem musel použít oscilátor kitu, nikoli integrovaný v mikroprocesoru, protože mikroprocesor obsahuje nepříliš přesný oscilátor, a při

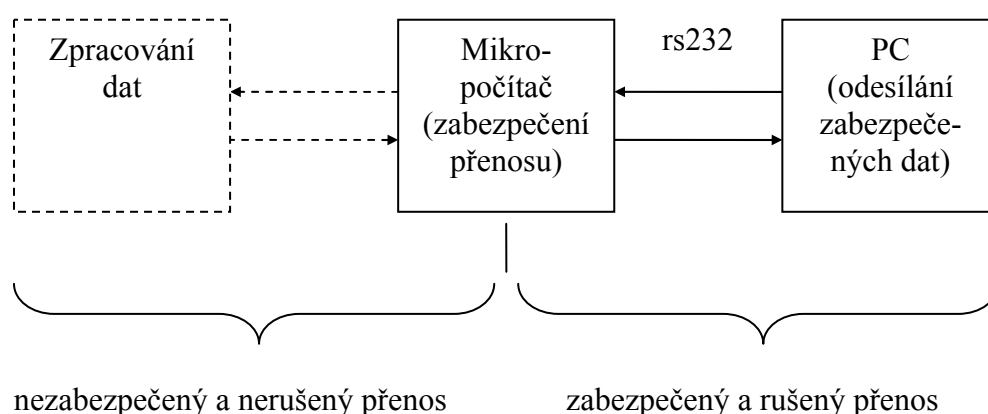
komunikaci s pc by docházelo ke špatné synchronizaci přenosu. Kit má také integrován napěťový převodník společně s MAX202CSE, který převádí sériové rozhraní USART (Universal Synchronous Asynchronous Receiver Transmitter) mikroprocesoru (0-5V) na hodnoty sériového rozhraní rs232 (viz. kapitola 5.1). Kit obsahuje dvě rozhraní rs232. Pomocí jednoho se provádí programování kódu do mikropočítače přes rozhraní ISP. Druhé slouží pro samotnou komunikaci mikroprocesoru, například právě s počítačem.

Rozhraní ISP (In-System Programming) slouží pro naprogramování kódu do mikročipu, bez nutnosti jej vyjmout z desky, kde je osazen.

Při provozu kitu jsem použil oscilátor 3.686 MHz, který je integrován na desce. Z této frekvence lze pomocí 5.2 dovést dobu zpracování jednoho strojového cyklu.

$$t_{sc} = \frac{1}{f_{osc}} = \frac{1}{3,686 \cdot 10^6} = \underline{\underline{271 \mu s}} \quad (5.2)$$

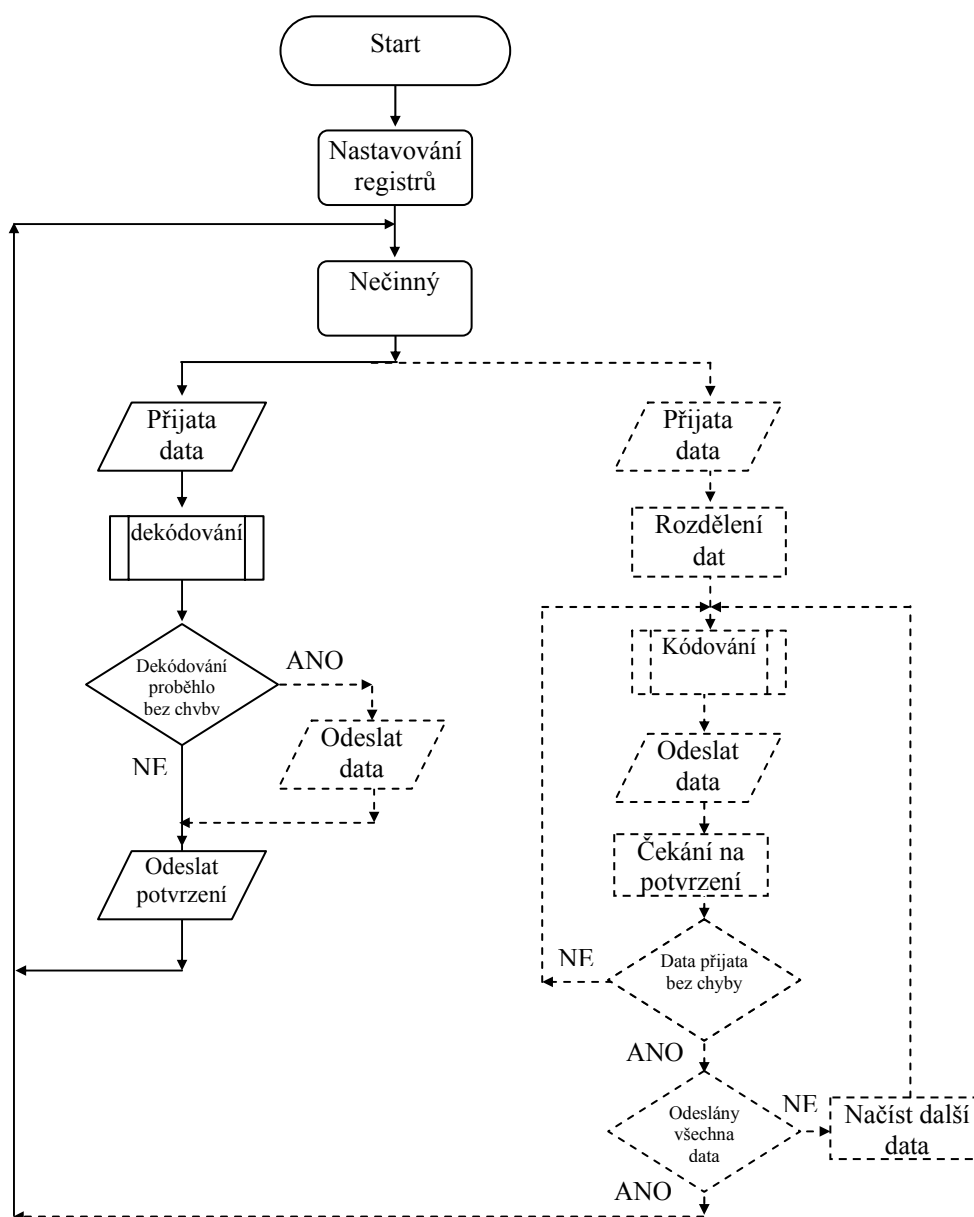
V případě praktického použití, by se systém mohl použít například v továrním prostředí s vysokým rušením. Zabezpečená sériová linka by sloužila pro přenos dat mezi počítačem a přístrojem, elektronika v přístroji už by byla v odstíněném prostředí, a tudíž by nebylo potřeba přenos zabezpečovat. Blokové schéma je znázorněno na obr. č. 19. Blok a rozhraní, který následně zpracovává dekodovaná data je znázorněn čárkovaně.



Obr. č. 19: Zapojení mikropočítače do systému

5.5.1 Samotná funkce programu mikropočítače

Vývojový diagram programu, který je nahrán v mikropočítači, je na obr. č. 20. Funkce je velmi podobná programu v počítači. Po zapnutí (respektive přivedení napájení) se provede počáteční nastavení stavových registrů. To zajistí nastavení např. typů portů (vstupní/výstupní), jejich počáteční hodnoty (1 nebo 0), rychlost přenosu dat rozhraní USART, atd. Následně přejde program do stavu nečinný, a čeká na příchod dat. Data jsou detekována příchodem synchronizačního bajtu. Poté z přijaté hlavičky jsou načteny parametry použitého kódu. Následují již zabezpečená data. Po dekodování a opravení chyb by v případě použití zapojení do systému následoval proces, který by data odesílal jiným portem k dalšímu



Obr. č. 20: Vývojový diagram programu v mikropočítači

zpracování (znázorněno čárkovanou čarou). Dále pak mikroprocesor sestaví a odešle potvrzení, které umožní vysílajícímu PC posílat další data. V případě výskytu neopravitelné chyby by data nebyla předávána, ale bylo by odesláno záporné potvrzení, které požaduje opětovné vyslání dat. Kdyby byl systém zapojen do přenosového systému, byla by také možnost, že přijdou nějaká data pro odeslání do počítače. V tom případě by se data načetla a po částech by se postupně zabezpečeně odesílala. Tento případ, kdy jsou data odesílána je zobrazen na obr. č. 20 (čárkovaná smyčka v pravé části obrázku).

Abychom měli alespoň nějakou kontrolu, o tom že se data opravdu do kitu přenášejí a jsou dekodována správně, je z každého rámce první bajt zobrazen pomocí LED diod.

5.6 Porovnání jednotlivých kódů

Testování jednotlivých kódů jsem prováděl trojí, pokaždé se jedná o závislost doby přenosu na velikosti souboru. Výsledky testů jsou uvedeny v tab. č. 12, graficky jsou znázorněny na obr. č. 21. Přenos se uskutečňoval na bezchybném vedení, přenášeny byly 3 soubory o velikostech 192, 420 a 2100 bajtů. Který kód je nejlepší nelze jednoznačně říci, neboť závisí na specifických požadavcích uživatele a chybovosti linky.

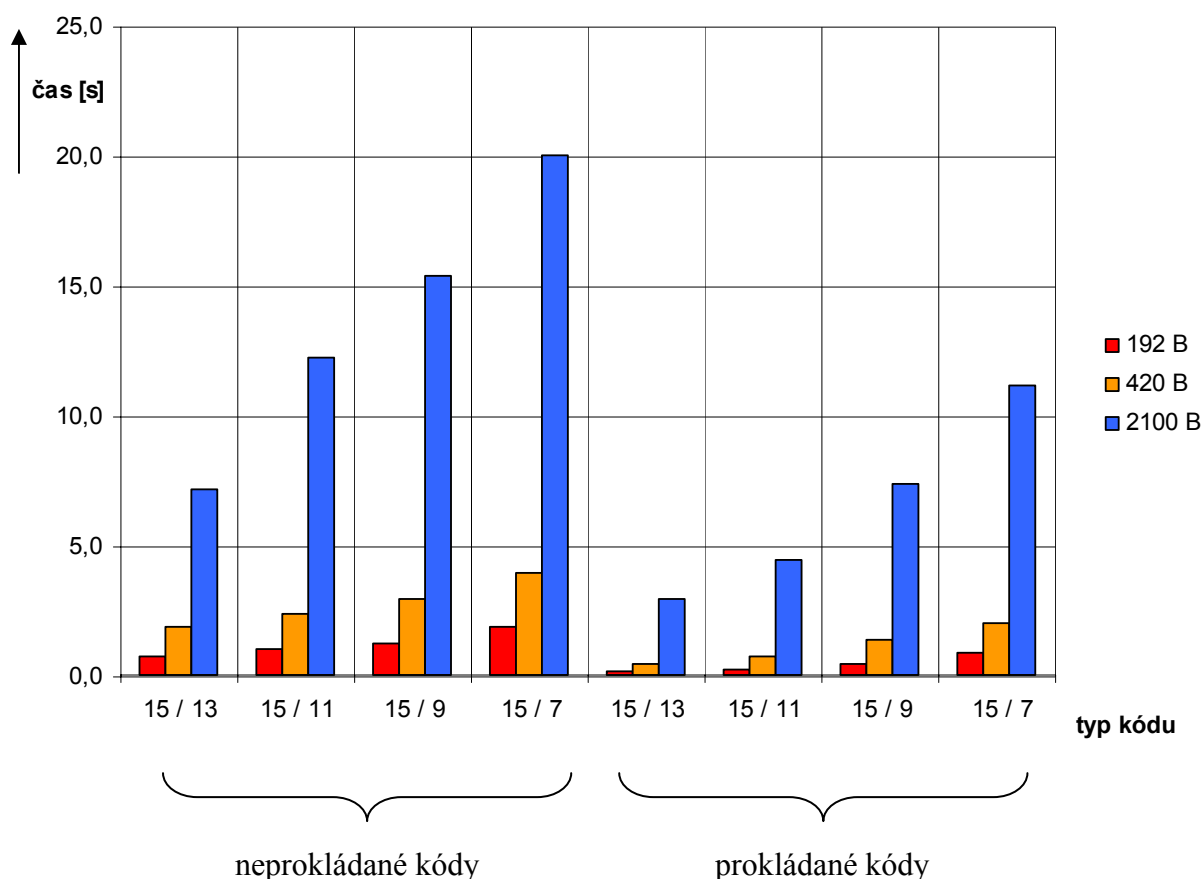
Tab. č. 6: Výsledky testování přenosem souborů

pořadí	kód	prokládání	Přenosová rychlost dat [kB/s]	doba přenosu souboru [s]		
				192	420	2100
1	15 / 13	ne	6,912	0,8	1,9	7,2
2	15 / 11	ne	5,760	1,1	2,4	12,3
3	15 / 9	ne	4,608	1,3	3,0	15,4
4	15 / 7	ne	3,456	1,9	4,0	20,1
5	15 / 13	ano	8,378	0,2	0,5	3,0
6	15 / 11	ano	6,922	0,3	0,8	4,5
7	15 / 9	ano	5,585	0,5	1,4	7,4
8	15 / 7	ano	4,189	0,9	2,1	11,2

Jednoznačně ale vyplývá, že je vhodnější použít prokládané kódy, tedy kódy číslo 5 až 8, protože v tomto případě dosahují vyšších přenosových rychlostí a větší odolnost proti shlukům chyb. Důvodů je několik:

- zpoždění vzniklé prokládáním je malé (viz. vzorec 5.3 , hodnota 8855 je počet strojových cyklů, potřebných pro provedení kódu prokládání)
- paměťové nároky na prokládací paměť jsou minimální, mikroprocesory dnes obsahují dostatečně velké paměti, za velmi nízké pořizovací náklady

- tím že se při prokládaném přenosu spojí osm rámců za sebe, aby se mohly proložit, a následně se odešlou jako jeden rámec, sníží se sedmkrát počet režijních dat (synchronizace a záhlaví) a tím se docílí vyšší přenosové rychlosti (viz. vzorec 5.6)
- potvrzení přijatých prokládaných dat se posílá pouze jedno namísto osmi (ušetří se čas $7 \times t_{PRN1}$)



Obr. č. 21: Graf zobrazující dobu přenosu jednotlivých souborů použitými kódy

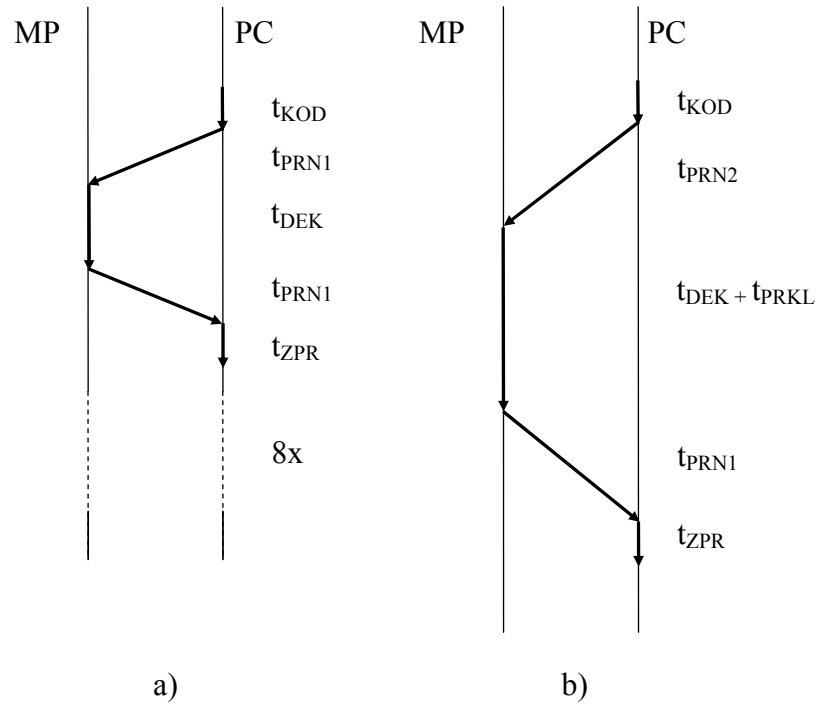
$$t_{PRKL} = \frac{1}{f_{OSC}} \cdot 8855 = \frac{1}{3,686 \cdot 10^6} \cdot 8855 = 2,40ms \quad (5.3)$$

$$t_{PRN1} = t_{\text{přenosu_jednoho_bajtu}} \cdot 10 = 0,0868 \cdot 10 = 0,868ms \quad (5.4)$$

$$t_{PRN2} = t_{\text{přenosu_jednoho_bajtu}} \cdot 66 = 0,0868 \cdot 66 = 5,7288ms \quad (5.5)$$

$$t_{\text{ušetřeny_pri_přenosu}} = t_{PRN1} \cdot 8 - t_{PRN2} = 0,868 \cdot 8 - 5,7288 = 1,2152ms \quad (5.6)$$

Podrobněji popisuje časové rozdíly obr. č. 22. Doba kódování t_{KOD} a t_{ZPR} lze zanedbat, protože je prováděna v počítači mnohonásobně rychleji, než jakým pracuje zbytek přenosového systému. Uvažujeme tedy dobu dekodování t_{DEK} , dobu přenosu neprokládaného rámce t_{PRN1} a dobu přenosu prokládaného rámce t_{PRN2} . U prokládaného přenosu je k t_{DEK} zahrnuta také doba t_{PRKL} , potřebná pro proces prokládání (viz. vzorec 5.3).



Obr. č. 22: Znázornění dob přenosů pro přenesení stejného množství dat u a) neprokládaného systému b) u prokládaného systému

Vzorec 5.7 vyjadřuje dobu pro přenesení osmi rámců s daty, pro libovolný typ kódu. Druhý případ, kdy je použito prokládání, a je přeneseno stejné množství dat, vyjadřuje vzorec 5.8.

$$t_{neprokladane} = 8 \cdot (t_{PRN1} + t_{DEK} + t_{PRN1}) = 8 \cdot t_{DEK} + 16 \cdot t_{PRN1} = 8 \cdot t_{DEK} + 13,888 \quad (5.7)$$

$$t_{prokladane} = t_{PRN2} + t_{PRKL} + 8 \cdot t_{DEK} + t_{PRN1} = 5,7288 + 2,40 + t_{DEK} + 0,868 = 8,997 + 8 \cdot t_{DEK} \quad (5.8)$$

Poslední vzorec 5.9 vyjadřuje t_{rozdil} , tedy o kolik je rychlejší prokládaný přenos oproti neprokládanému.

$$t_{rozdil} = t_{neprokladane} - t_{prokladane} = 8 \cdot t_{DEK} + 13,888 - 8 \cdot t_{DEK} - 8,997 = \underline{\underline{4,891ms}} \quad (5.9)$$

Jak již bylo zmíněno výše, prokládané kódy zabezpečují v našem případě proti osmkrát větším shlukům chyb, a vlivem některých úspor jsou i následně rychlejší. Jejich nevýhodou může být například stav, kdy se vyskytne v rámci tak velká chyba, kterou by nebyl dekodér schopen opravit. V tom případě by se musel vysílat celý rámec znovu, což by při použití neprokládaného kódu znamenalo dřívější odhalení chyby a menší množství dat, které se bude opět vysílat.

Původním záměrem bylo realizovat kód s délkou symbolu 8 bitů, aby význam jednoho symbolu odpovídal jednomu znaku v ascii tabulce. Ovšem při naprogramování dekodéru se schopností opravit 10 a více chyb nastaly problémy s velikostí paměti u mikropočítače. Matice syndromů, i přes to že obsahovala hodnoty char a nikoli int, si vyžadovala velké nároky na paměť. Takovéto mikroprocesory sice v nabídce existují, ale jsou buď extrémně drahé, nebo nejsou běžně dostupné v ČR, nebo nejsou v klasickém pouzdře, ale v provedení TQPF. Toto provedení nelze osadit do vývojového kitu.

Jelikož můj počítač disponuje pouze jedním rozhraním COM, musel jsem si vypomoci redukcí USB/COM. Jelikož ovšem na tento převodník neexistuje žádná specifikace či norma, nelze zaručit 100%-ní funkčnost tohoto rozhraní. Moje osobní zkušenost je taková, že naprogramovat mikropočítač přes toto rozhraní šlo bez problémů, ovšem při samotné komunikaci počítače s kitem (při zabezpečeném přenosu dat), docházelo k chybám, nebo k úplné ztrátě komunikace.

6. Závěr

Reed Solomonovy kódy byly vyvinuty v roce 1960, a od té doby byly ještě zdokonaleny. Dnes mají široké využití pro jejich výbornou zabezpečovací schopnost. Nejobtížnějším na celém kódu je dekodování, tzn. proces zjištění a opravení chyby. Proces je rozdělen do několika fází, a každá z nich se dá realizovat několika algoritmy. Záleží pouze na našich požadavcích a technických možnostech. Berlekamp-Massey algoritmus, používaný pro nalezení chybového polynomu, je velice efektivní, ale také výpočetně náročný. Jeho princip graficky znázorňuje program vytvořený v prostředí Matlab. Pomocí tohoto programu můžeme ověřit vlastnosti různých kódů. Jako praktickou implementaci tohoto protichybového kódu jsem použil zabezpečení komunikace přes rozhraní RS232. Komunikace probíhá mezi programem na počítači a mikropočítačem firmy Atmel. Použil jsem několik typů kódů, pokaždé s délkou zabezpečeného slova 15 znaků, lze také použít osminásobné prokládání. Nejrychlejším kódem je 15 / 13 s prokládáním, kdy se data přenáší rychlostí 8,378 kB/s. Kódem schopným opravit největší shluk chyb je 15 / 7 s prokládáním, který má přenosovou rychlost dat 4,189 kB/s. Tento kód dokáže opravit shluky chyb až o velikosti 32 z celkem 112 znaků. V tomto případě se jeví prokládané kódy jako rychlejší, protože doba potřebná pro realizaci prokládání je několikanásobně menší, než doba ušetřená opětovným nepřepřenáním režijních dat. Ta vznikají tím, že při prokládání přenášíme hlavičku a synchronizační značku pouze jednou namísto osmkrát.

Použitá literatura

- [1] HANZO L., LIEW T. H. , *The Coding, Turbo Equilisation and Space-Time Cosiny for transmission over fading channels*, 2004. John Wiley & Sons Ltd., ISBN: 0-470-84726-3
- [2] LIN S., COSTELLO J., DANIEL J., *Error Control Coding: Fundamentals and Applications*, second edition, Prentice Hall: Englewood Cliffs, NJ, 2005 ISBN: 0-13-042672-5
- [3] SKALAR B., *Digital Communications, Fundamentals and applications*, Prentice-Hall, 2003, ISBN 0-13-084788-7
- [4] ZAPLATÍLEK K., DOŇAR B, *Matlab : pro začátečníky.* , 1. vydání , Praha : BEN, 2003, ISBN 80-7300-095-4
- [5] ZAPLATÍLEK K., DOŇAR B, *Matlab : tvorba uživatelských aplikací.* , 1. vydání , Praha : BEN, 2004, ISBN 80-7300-133-0.
- [6] KADLEC V., *Učíme se programovat v Borland C++Builder a jazyce C++*, 1. vydání , Praha : Computer Press, 2002, ISBN 80-7226-550-4
- [7] ECKEL B., *Myslíme v jazyku C++*, 1. vydání , Praha : Grada Publishing, 2000, ISBN 80-247-9009-2
- [8] BURKHARD K. , *Využití rozhraní PC pro Windows*, 1. vydání , Praha : BEN, 2002, ISBN 80-86167-13-5
- [9] FIALA F. , *Sériový port počítače PC v laboratorních úlohách*, 1. vydání , Ústí nad Labem : UNIVERZITA J.E. PURKYNĚ, 2003, ISBN 80-7044-504-1
- [10] VÁŇA V. , *Mikrokontroléry ATMEL AVR programování v jazyce C*, 1. vydání , Praha :BEN , 2003, ISBN 80-7300-102-0
- [11] FATOUŠEK D. , *C pro mikrokontrolery ATMEL AT89S52*, 1. vydání , Praha : BEN, 2007, ISBN 978-80-7300-215-9

Seznam použitých zkratek

Proměnné

a	– koeficienty polynomu $A(x)$
d	– odchylka
h	– hloubka prokládání
j	– počet vzniklých chyb
k	– počet vstupních symbolů (informace)
l	– délka LFSR
m	– počet bitů potřebných pro vyjádření jednoho slova
n	– počet výstupních symbolů (zabezpečené informace)
r	– stupeň rozšíření
t	– počet chyb, které jsme schopni opravit
t_{zp}	– doba zpoždění
t_{sc}	– doba jednoho strojového cyklu
f_{osc}	– frekvence oscilátoru
x	– hodnota signálového prvku
R	– informační poměr

Polynomy

$a(x)$	– mnohočlen prvků Galoisova tělesa
$e(x)$	– chybový mnohočlen (chyby na vedení)
$f(x)$	– zakódovaná zpráva
$f'(x)$	– zakódovaná zpráva s chybami
$g(x)$	– vytvářecí mnohočlen
$q(x)$	– polynom (výsledek po dělení)
$r(x)$	– polynom (zbytek po dělení)
$A(x)$	– pomocný chybový polynom
$E(x)$	– rozhodovací polynom
$L(x)$	– chybový polynom (hledání chyb BM algoritmem)
$M(x)$	– polynom určující velikost chyb
$P(x)$	– polynom určující pozici chyb
$S(x)$	– polynom syndromů
$T(x)$	– dočasný chybový polynom

Zkratky

LSB – nejméně významný bit

MSB – nejvíce významný bit

RXD – přijímač

TXD – vysílač

Přílohy

Příloha 1: obsah přiloženého CD

Název souboru (složky)	Obsah souboru (složky)
ctime.txt	soubor popisující obsah CD
rs.exe	spustitelný program pro počítač
text.pdf	text diplomové práce
matlab_aplikace	složka obsahující aplikaci v programu Matlab
projekt_c_pocitac	složka obsahující celý projekt programu pro počítač
projekt_c_atmel	složka obsahující celý projekt programu pro mikropočítač