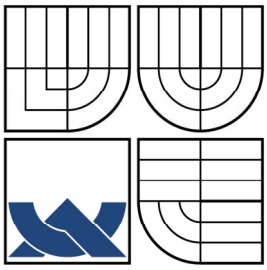
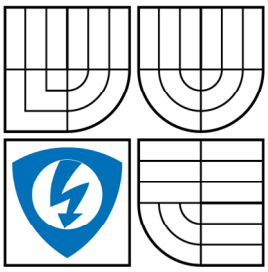


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATION

ZABEZPEČENÍ PŘENOSU DAT BCH KÓDY

ERROR PROTECTION OF DATA TRANSMISSION USING BCH CODES

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE
AUTHOR
BC. JAROSLAV KAŠPAR

VEDOUCÍ PRÁCE
SUPERVISOR
DOC. ING. KAREL NĚMEC, CSC.

BRNO 2008

Oficiální zadání práce

První list licenční smlouvy

Druhý list licenční smlouvy

Anotace

Diplomová práce Zabezpečení přenosu dat BCH kódy se zabývá významnou skupinou cyklických kódů, které jsou schopny zabezpečit binární data před nezávislými chybami. Bose, Chaudhuri a Hocquenghem (BCH) kódy využívají Galoisova tělesa. Kódování je stejné jako u cyklických kódů a může být použit lineární zpětnovazební posuvný registr. Dekódování je složitější a lze využít několik algoritmů - v této práci jsou zmíněny dva: Petersonův a Berlekam-Massey. Cílem této práce je nalézt BCH kód opravující až $t = 6$ nezávislých chyb v posloupnosti $n = 150$ bitů, poté prostudovat možné realizace a podle kritérií vybrat a navrhnout kodek vybraného kódu. Tuto práci lze pomyslně rozdělit do tří částí: nejprve je uveden teoretický popis kódování a dekódování BCH kódem, následuje výběr kódu a realizace (byl vybrán kód BCH (63, 30) a realizace pomocí FPGA obvodu). V poslední části je uveden návrh kodéru a dekodéru BCH kódu.

Klíčová slova

BCH kód, kodér, dekodér, kodek, binární, oprava chyb, přenos dat, dekódování, realizace, FEC, FPGA, protichybový kódový systém.

Abstract

The thesis Data transmission error-protection with BCH codes deals with a large class of random-error correcting cyclic codes which are able to protect binary data and can be used for example in data storages, high speed modems. Bose, Chaudhuri and Hocquenghem (BCH) codes operate over algebraic structures called Galois fields. The BCH encoding is the same as cyclic encoding and can be done with linear feedback shift register but decoding is more complex and can be done with different algorithms - in this thesis there are two algorithms for decoding Peterson and Berlekam-Massey mentioned.

The aim of this thesis is to find BCH code which is able to correct $t = 6$ independent errors in up to data sequence $n = 150$ bits, then peruse possible realizations of the codecs and set criteria for the best realization, then design and test functionality of the realization.

This thesis is split into three main parts. In the first part there are encoding and decoding methods of the BCH code generally described. The second part deals with selecting of the realization and code. There was chosen BCH (63,30) code and realization with FPGA chip. In the last part is described design of BCH encoder and decoder in VHDL language and this source code is compiled in the Altera design software.

Keywords

BCH code, encoder, decoder, codec, binary, error correction, data transmission, decoding, FEC, FPGA, implementation.

KAŠPAR, J. *Zabezpečení přenosu dat BCH kódy.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 55 s. Vedoucí diplomové práce doc. Ing. Karel Němec, CSc.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma "**Zabezpečení přenosu dat BCH kódy**" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a 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 10. května 2008 ……...

Poděkování

Děkuji vedoucímu diplomové práce, Doc. Ing. Karlu NĚMCOVI, CSc., za velmi užitečnou metodickou, pedagogickou a odbornou pomoc a další rady při zpracování mé diplomové práce.

V Brně dne 10. května 2008 ……

Obsah

1. Úvod ... 10
1.1 Cíl práce ... 11
1.2 Základní struktura komunikačního systému ... 12
1.3 Způsoby detekce a opravy chyb ... 13
2. Obecný popis BCH kódů ... 14
2.1 Základní pojmy o BCH kódech ... 14
2.2 Základní pojmy z lineární algebry ... 16
3. Simulace BCH kódů v prostředí Matlab - Simulink ... 18
3.1 Návrh přenosového kanálu ... 18
3.2 Sestavení a simulace ... 19
3.3 Výsledky simulace BCH kódů v Matlabu ... 22
4. Výběr a sestavení BCH kódu podle zadání ... 23
4.1 Výběr kvalitního BCH kódu ... 23
4.2 Sestavení vytvářecího mnohočlenu *g(x)* ... 24
5. Kodér BCH kódu ... 26
5.1 Realizace pomocí posuvných registrů ... 27
6. Dekodér BCH kódu ... 30
6.1 Kontrola správnosti přijatého kódového slova ... 31
6.2 Určení chybového mnohočlenu – lokátoru chyb ... 31
6.3 Určení pozice chyb a jejich oprava ... 33
7. Možné realizace BCH kodeku ... 34
7.1 Soubor realizací BCH kodeku ... 34
7.2 Výběr realizace - kritéria ... 39
8. Návrh realizace protichybového kodeku ... 41
8.1 Prostředky a postupy pro programování FPGA obvodu ... 41
8.2 Návrh kodéru ... 43
8.3 Návrh dekodéru ... 45
8.4 Projekt v programu Quartus II ... 46
9. Závěr ... 48
Použitá literatura ... 50
Seznam obrázků ... 52
Seznam tabulek ... 53
Seznam zkratek ... 54

1. Úvod

Jedna ze základních vlastností nejrůznějších druhů informačních systému by měla být schopnost efektivně a kvalitně komunikovat. Pod pojmem komunikace si lze představit předávání zpráv mezi informačními systémy, nacházejícími se na rozdílných místech. Za určitý druh komunikace lze považovat i ukládání zpráv do paměti a jejich následné čtení. V obou případech je však nutné přenášet data přes nějaké fyzikální prostředí (přenosovým kanálem), které bývá zpravidla zdrojem rušení. Toto rušení způsobuje vznik chyb, které způsobí snížení kvality přenášených dat nebo tyto data naprosto znehodnotí. V dnešní době „komunikační revoluce“, kdy se rozvíjí komunikace přes mobilní telefonní sítě, satelity a počítačové sítě (Internet) jsou kladeny čím dál tím větší nároky na přenosové rychlosti a je nutné, aby tato komunikace byla co možná nejvíce spolehlivá a efektivní. Ne vždy jsou přenosové podmínky ideální (hlavě bezdrátový přenos a metalické vedení), a proto je nutné tyto přenosy nějakým způsobem zabezpečit. Pro představu: pokud bychom měli 1Gb/s přenosový kanál s 0,001% chybovostí, znamenalo by to, že ztratíme 10 000 bitů za sekundu. Podle [1] obsahují průměrné noviny 1 000 slov, to odpovídá 42 000 bitů, takže bychom přišli přibližně každé čtyři sekundy o jedny noviny. Je vidět, že i malá chybovost je nepříjemná při přenosu dat přes vysokorychlostní přenosové kanály.

Dr. Claude Shannon z Bell Telephone Laboratoriem již v roce 1948, říká, že i ve špatném přenosovém kanálu lze zakódovat přenášenou zprávu tak, aby byly chyby při přenosu potlačeny na minimum [1]. Shannon došel k závěru, pokud náš informační zdroj bude předávat informace menší rychlostí, než je náš komunikační kanál schopen přenést potom můžeme přidat speciální informaci a snížit tak pravděpodobnost chybného přenosu dat.

Dnešní technologický rozvoj a hlavně rozboj v oblasti číslicové techniky poskytuje možnosti pro implementaci náročnějších algoritmu na protichybové zabezpečení dat, než které bylo možné realizovat v letech minulých. Otevírají se tak možnosti pro praktické ověření postupů popsaných dříve teoretický a které v minulosti pro nedostatek potřebného výkonu nebylo možné prakticky realizovat. Jednou z mnoha oblastí, kde se dá vzrůstající výpočetní výkon uplatnit je právě oblast komunikací a digitální uložiště dat.

Vysvětlení některých nejzákladnějších pojmů:

- Komunikační kanál – můžeme si ho představit jako trubici určitého typu – např. pokud chceme přenášet text nebo obraz může to být počítačová síť Internet,

- Kapacita kanálu – maximální množství dat, které můžeme přenést skrz komunikační kanál za určitý časový úsek – např. pro modemovou linku 56kbit/s,

- Informační zdroj – např. soubor přenášený z FTP serveru. Server dokáže zpřístupnit data nějakou rychlostí (Mbit/s), většinou je tato rychlost větší, něž je kapacita kanálu,

- Šum – má vliv na změny přenášených dat. Existuje několik druhů šumu.

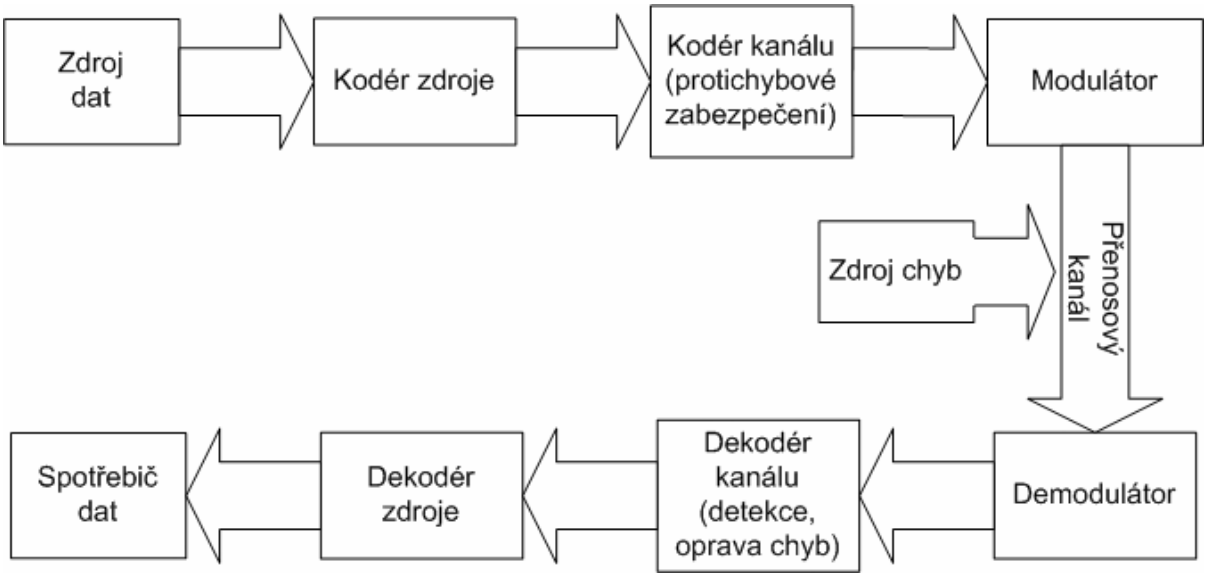
1.1 Cíl práce

Cílem této práce je zjistit a popsat možnosti BCH (Bose-Chaudhuri-Hocquenghem) kódů pro opravu nezávislých chyb v digitálním přenosu dat. Výsledný kód by měl mít schopnost opravit až $t = 6$ nezávislých chyb v celkovém úseku bitového toku délky maximálně $n = 150$ bitů. Dále sestavit kritéria pro výběr vhodného kodeku, který bude splňovat podmínky zadání a dále sestavit soubor možných realizací tohoto vybraného kodeku. V poslední řadě vybrat a provést návrh vhodné realizace a ověřit její funkční schopnosti.

Nejprve bude v této práci uveden popis základních principů a parametrů týkajících se BCH kódů. V následující kapitole bude uveden způsob jakým lze najít a sestavit vytvářecí mnohočlen BCH kódu, který splňuje požadované zadání práce. Následuje matematický popis kodéru a dekodéru BCH kódu, soubor možných realizací a výběr nejvhodnější varianty a návrh vybrané realizace kodeku.
Osnova:

1. Úvod, výběr a sestavení vytvářecího mnohočlenu BCH kódu,
2. Matematický popis kodéru a dekodéru BCH kódu,
3. Možné realizace a výběr nejvhodnější varianty,
4. Návrh vybrané varianty.

1.2 Základní struktura komunikačního systému



Obr. 1-1 Struktura komunikačního kanálu

Pro názornost uvádím schéma komunikačního systému s popisem jednotlivých bloků. Tato část by měla poskytnout základní přehled a úvod do částí, které budou popisovány podrobněji dále.

Prvním blokem v řetězci je „Zdroj dat“. Zdroj dat může být např. člověk nebo stroj, který potřebuje komunikovat se spotřebičem dat na opačném konci řetězce.

Dalším blokem je „Kodér zdroje“, který převádí data od zdroje na posloupnost binárních znaků. Tento blok si můžeme představit např. jako A/D převodník. Správně navržený kodér zdroje odstraní ze zpráv nadbytečnou redundanci (vede ke zvýšení efektivity komunikace) a zajistí, aby data na přijímací straně byly jednoznačně dekódovány [5].

„Kodér kanálu“ převádí nezabezpečenou posloupnost binárních znaků na zabezpečenou posloupnost znaků – kódové slovo [5]. Zavádí do zprávy určitou redundanci, záleží na zvoleném kódovém systému. Na přijímací straně potom můžeme v „Dekodéru kanálu“ detekovat (detekční kódy) nebo dokonce opravit (korekční kódy) přijatá data a zefektivnit tak přenos dat. Velice záleží na vhodném výběru kodeku pro zabezpečení dat v závislosti na prostředí, ve kterém tyto data budou přenášena (přenosové médium). V této práci se zaměřím na využití BCH kódů, které patří do skupiny korekčních kódů pro zabezpečení binární posloupnosti znaků.

Dalším blokem je „modulátor“, který vstupní proud dat přizpůsobuje pro přenos přes přenosový kanál nebo pro možnost uložení na paměťové médium.
Přenos dat přes přenosový kanál (prostředí) je ovlivněno rušením. Nejběžnějším přenosovým prostředím je metalické vedení nebo vzduch. Každé prostředí může mít charakteristické vlastnosti rušení, a proto není jednoduché navrhnout protichybové zabezpečení dat tak, aby byl přenos co nejefektivnější.

1.3 Způsoby detekce a opravy chyb

Pro zajištění bezchybného přenosu dat přes komunikační kanál existují dvě hlavní metody:

- ARQ – Automatic Repeat reQest – na přijímací straně dochází pouze k detekci, zda došlo k poškození dat během přenosu a na základě této detekce se rozhoduje, zda je nutné opakovat přenos požadovaných dat. Výhodou je, že nemusíme implementovat složité korekční algoritmy na přijímací straně, protože detekce je jednodušší než korekce dat. Nevýhodou je, že musíme mít možnost obousměrné komunikace – musíme zaslat požadavek od přijímací strany a také se zvyšuje počet přenesených dat (požadavek od přijímače + opakované odeslání dat),

- FEC – Forward Error Correction – na přijímací straně dochází k detekci i opravě chyb vzniklých při přenosu dat přes komunikační kanál. Na přijímací straně je implementován korekční algoritmus. Tento způsob je možné použít v případě, kdy není možné komunikovat obousměrně nebo chceme snížit množství přenesených dat. V dalším textu se budu věnovat skupinou korekčních kódů zvanou BCH.

Výběr konkrétní metody zabezpečení přenášených dat závisí na možnostech přenosového systému a na efektivitě, které chceme při přenosu dat dosáhnout. Možná je také kombinace obou uvedených způsobů.

2. Obecný popis BCH kódů

2.1 Základní pojmy o BCH kódech

Skupina BCH kódů je pojmenována podle iniciál objevitelů Bose – Chaudhuri – Hocquenghem. Řadí se mezi lineární blokové cyklické kódy a jsou postaveny na Hammingových kódech, ale oproti těmto kódům umožňují opravit více chyb. BCH kódy byly objeveny panem Hocquenghem v roce 1959 a nezávisle Bose, Chaudhurim v roce 1960 [5]. Konstrukce BCH kódů zaujímá důležité místo jak v teorii, tak v praxi. BCH kódy mohou být jednoduše implementovány v digitálních zařízeních, protože pracují s daty v binární podobě. Mezi jejich dobré vlastnosti patří možnost volitelnosti parametrů (délka kódového slova, počet opravitelných chyb), dobrý vztah mezi počtem informačních znaků a počtem opravovaných chyb. První algoritmus pro dekódování byl navržen W. W. Petersonem v roce 1960. Následně po něm byly nalezeny další dekódovací algoritmy panem Berlekampem, Chienem, Forneyem [7].
Realizace kodéru BCH kódu je poměrně jednoduchá (vychází ze zapojení lineárního posuvného registru), avšak bohužel realizace dekodéru je poměrně složitější a vyžaduje implementaci několika matematických postupů.

Speciální třídou BCH kódů jsou RS kódy (Reed – Solomonovy kódy). RS kódy slouží k zabezpečení nebinárních posloupností a pracují se symboly (použití např. v modemech ADSL).

Rozsah použití BCH kódů spadá do mnoha oblastí, kde je potřeba pracovat s binárními daty, nejedná se pouze pro zabezpečení datových přenosových kanálů. Mezi příklady využití BCH kódů patří zálohovací média (CD, DVD, disková pole), bezdrátové a satelitní komunikace, digitální televize (standard DVB), vysokorychlostní modemy atd.

Zařazení BCH kódů:

Jak bylo již uvedeno, BCH kódy patří do skupiny lineárních blokových cyklických kódů, pracujících s binárními daty tj. 0 nebo 1 a jsou účinné jako protichybové zabezpečení bitových toků v přenosových systémech, ve kterých nedochází ke shlukům chyby.

Význam jednotlivých pojmů:

- Lineární kód

Libovolná kódová kombinace lineárního kódu může být odvozena lineární kombinací některých jeho dalších kombinací. Lineární kódy bývají zadány vytvářecí maticí G o *k* řádcích a *n* sloupcích (označují se jako *(n, k)* kódy). Jako řádky jsou použity ty libovolné kombinace, které jsou vzájemně lineárně nezávislé (žádnou lineární operací mezi libovolnými řádky matice G nevznikne jiný řádek této matice).

- Blokový kód

Nezabezpečený datový tok je rozdělen do bloků o stejné bitové délce *k*, tyto bloky jsou poté postupně kódovány jeden po druhém. Výsledkem je kódové slovo délky *n* (je přidána určitá redundance – zabezpečení), platí $n > k$. Výstupní zakódované zprávy mají stejnou délku. Detekce a korekce chyb závisí pouze na konkrétním bloku a ne na blocích předcházejících nebo následujících.

- Cyklický kód

Jsou to lineární kódy *(n, k)* obsahující kódové slovo **g***(x),* které jako mnohočlen má stupeň *(n - k).* Všechna další kódová slova jsou násobky mnohočlenu **g***(x).* Mnohočlen **g***(x)* se nazývá generující mnohočlen cyklického kódu. Tento mnohočlen dělí beze zbytku dvojčlen $(x^n - 1)$. V kódové kombinaci délky *n* prvků mají cyklické kódy na prvních *k* místech prvky nezabezpečené zprávy a na zbývajících *r* místech zabezpečovací prvky.

Cyklická vlastnost říká, že pokud $[f_0, f_1, f_2, \ldots, f_{n-1}]$ je kódové slovo lineárního blokového cyklického *(n, k)* kódu, jsou s ním následující slova také kódová:

$[f_{n-1}, f_0, f_1, \ldots, f_{n-2}]$,

$[f_{n-2}, f_{n-1}, \ldots, f_{n-3}]$, (2.1)

.

.$[f_1, f_2, \ldots, f_{n-1}, f_0]$.

Kódové slovo $[f_0, f_1, \ldots, f_{n-1}]$ cyklického kódu může být reprezentováno také jako polynom

$$\mathbf{f}(x) = f_0 + f_1 x + ... + f_{n-1} x^{n-1} \quad (2.2)$$

2.2 Základní pojmy z lineární algebry

V této kapitole vycházím z informací uvedených v literatuře [3], [4] a [7].

Operace spojené se zabezpečením nezabezpečených signálových prvků BCH kódy se provádí pomocí prvků Galoisova tělesa GF(p^r), kde p je základ číselné soustavy (odpovídá počtu stavů signálu – pro dvojstavový signál $p = 2$) a r je stupeň rozšíření Galoisova tělesa. Galoisovo těleso vznikne rozšířením konečného tělesa Z_p a je tvořeno konečným počtem prvků $n = p^r$, což odpovídá celkovému počtu kódových prvků v mnohočlenu zabezpečené zprávy.

- **Konečné těleso Zp**

Konečné těleso je algebraická struktura určená počtem svých prvků a tím je mocnina prvočísla *p.* Tato struktura je tvořena zbytky po dělení celých kladných čísel prvočíslem *p*. Prvočíslo *p* určuje základ číselné soustavy. Pro dvoustavový signál se používá konečné těleso Z_2 tvořené množinou prvků (0, 1). Pro každé prvočíslo *p* definujeme na množině operace sčítání „+“ a násobení „*“. Operace součtu je chápána jako exclusive or (XOR) a operace násobení jako logický součin (AND). Tyto operace prováděné nad konečným tělesem splňují asociativní, komutativní a distributivní zákony.

Tab. 2-1 AND a XOR v tělese Z_2

XOR +	**0**	**1**
0	0	1
1	1	0

AND *	**0**	**1**
0	0	0
1	0	1

- **Galoisovo těleso GF(p^R)**

Jedná se vlastně o konečné těleso rozšířené stupněm *r*. Rozšířením vznikne množina vektorů o *r* prvcích, kde každý prvek je ze Zp. Celkový počet prvků Galoisova tělesa je p^r. Pro binární BCH kódy se používá GF (2^r), které vznikne rozšířením tělesa $Z_{2.}$

Galoisovo těleso lze vyjádřit pomocí zbytků po dělení mnohočlenů ze Z_2 určitým mnohočlenem (vytvářecím mnohočlenem), pro který platí:

- Je nerozdělitelný v uvažovaném okruhu mnohočlenů
- Je stupně *r*
- Dělí bezezbytku $(x^n - 1)$ a žádný jiný dvojčlen nižšího stupně tohoto tvaru

Např.: Galoisovo těleso GF (2^4) je množina čtveřic prvků ze $Z_{2.}$ Určíme ji pomocí mnohočlenu z rozkladu dvojčlenu

$(2^{15} + 1) = (x + 1) . (x^2 + x + 1) . (x^4 + x^3 + x^2 + x + 1) . (x^4 + x + 1) . (x^4 + x^3 + 1)$

Při určování Galoisova tělesa se berou v úvahu pouze mnohočleny stupně *r* (v tomto případě *r* = 4). Zvolený mnohočlen se nazývá **vytvářecí mnohočlen GF (2^4)** a zpravidla se označuje $\mathbf{g}(x)_{GF.}$

Algebraické operace nad Galoisovým tělesem:

- Součet „ + “ se provádí jako sečítání po členech mnohočlenu $\mathbf{a}(x) \oplus \mathbf{b}(x) = \mathbf{c}(x)$, $c_i(x) = a_i(x) \oplus b_i(x)$, kde $\mathbf{a}(x)$, $\mathbf{b}(x)$, $\mathbf{c}(x)$ jsou mnohočleny vytvářející prvky GF (p^r).
- Násobení „ . “ je určeno jako zbytek po dělení součinu dvou prvků vytvářecím mnohočlenem GF (2^r), tj. $\mathbf{g}(x)_{GF}$.

V Galoisově tělese se nenulové prvky vyjadřují ve tvaru mocnin *α*. Mocnina určuje prvek buďto přímo, nebo prvek vyjádříme pomocí zbytku R[i] po dělení tohoto prvku s "velkým" exponentem vytvářecím mnohočlenem tělesa $G(x)_{GF}$ za předpokladu, že prvek a je primitivní. Tato podmínka je zajištěna, použijeme-li pro generování GF (2^r) primitivní mnohočlen.

3. Simulace BCH kódů v prostředí Matlab - Simulink

U BCH kódů je možnost nastavit různé parametry jako např. počet chyb, který má být opraven, délka kódového slova (bloku) a počet zabezpečovacích bitů. Protože v zadání je podmínka sestavit kritéria a porovnat jednotlivé kódy, které splňují zadání, rozhodl jsme se zkusit pro simulaci BCH kódů použít softwarový produkt firmy Mathworks Matlab Simulink, kde je možnost simulovat přenosový kanál a kodér a dekodér BCH kódů. Tato část by měla přinést přehled možností a měla by usnadnit rozhodnutí, který kód by měl být dále realizován. K testování BCH kódů jsem si zvolil Matlab Simulink. Tento software je snadno přístupný ve školních laboratořích a umožňuje snadné použití, proto mi tento způsob přišel jako nejefektivnější. Matlab lze použít i pro simulaci jiných kódů, lze tedy tento způsob použít obecně a porovnávat i různé typy kódů.

K dispozici jsem měl verzi **Matlab 7.0 (R14)**, pro BCH kódy je potřeba mít nainstalován „Communications Toolbox“.

3.1 Návrh přenosového kanálu

Protože BCH kódy patří do skupiny binárních blokových kódů, potřebujeme generátor binárních dat. Proto jsem vybral v Simulinku jako zdroj dat blok „Bernoulli Binary Generator“ s Bernoulliho distribucí.

Jako vhodný přenosový kanál lze použít blok „Binary Simetric Channel“, kterým můžeme simulovat přenosový kanál s výskytem chyb. Jedná se o binární přenosový kanál, chyby jsou generovány náhodně s pravděpodobností „Error probability“, kterou lze nastavit nebo přímo podle zadaného vstupního vektoru chyb. Délka tohoto vektoru chyb může být stejná jako vektor vstupních dat. Chyby jsou způsobeny inverzí přenášených bitů. To je poměrně výhodné, protože můžeme snadno měnit parametry a sledovat okamžité změny.

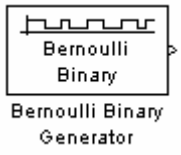
Další blok „Error rate calculation“ slouží k detekci a výpočtu chyb vzniklých přenosem a případně opravených nebo neopravených korekčním kódem. Blok má dva vstupy označeny Tx a Rx. Tx pro signál od zdroje (je brán jako referenční bez chyb) a Rx pro signál za přenosovým kanálem (tento signál je ovlivněn chybami vzniklými v přenosovém kanále). Výstupem tohoto bloku je vektor, který obsahuje informaci o

chybovosti kanálu BER (Bit error rate), počet chyb, a počet celkově přenesených bitů. Poslední blok slouží k zobrazování výsledku „Display“.

3.2 Sestavení a simulace

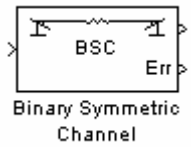
V další části jsem sestavil předem popsané zapojení a nastavil parametry pro otestování. Pro sestavení potřebujeme z knihovny Simulink Communication Toolbox následující bloky:

- Zdroj dat – „Bernoulli Binary Generator“



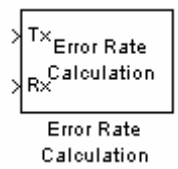
Parametry:
Probability of a zero = 0,5
Initial seed = 61
Sample time = 1
Samples per frame = 85 – záleží na zvoleném BCH kódu

- Binární přenosový kanál – „Binary Symmetric Channel“



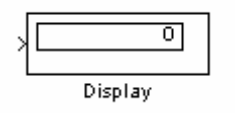
Parametry:
Error probability = pravděpodobnost výskytu chyby
Initial seed = 71
Output error vector = vzniklé chyby jsou ve výstupním vektoru

- Výpočet chyb – „Error Rate Calculation“

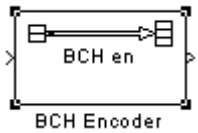


Tento blok provádí porovnání dat před přenosovým kanálem a za ním

- Blok pro zobrazení výpočtů – „Display“

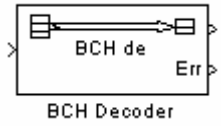


- Encodér BCH kódu – „BCH Encoder“

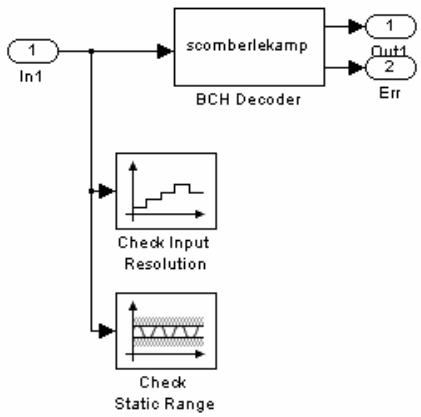


Parametry:
N = délka kódového slova
K = délka zabezpečované zprávy
N musí splňovat podmínku $N = 2^{M-1} \quad 3 \le M \le 9$ (3.1)
Vstupní data musí mít délku *k*.

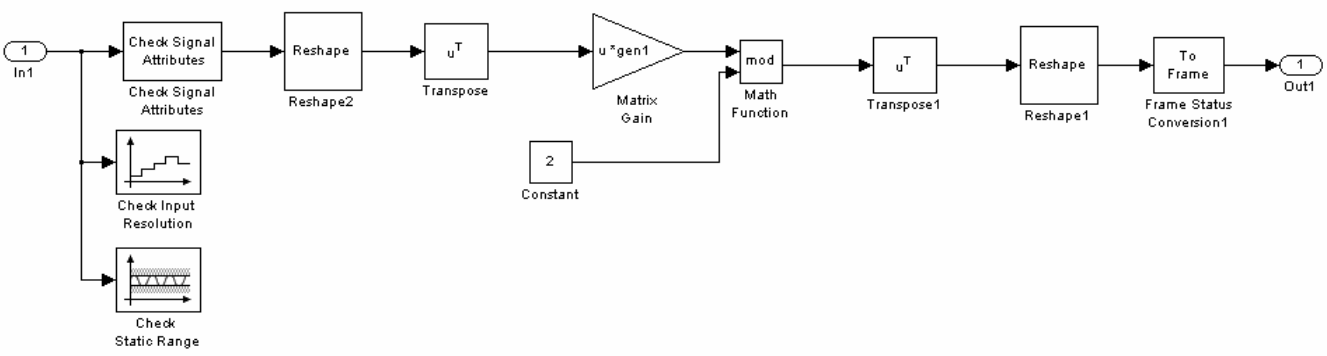
- Dekodér BCH kódu – „BCH Decoder“



n= délka kódového slova
k = délka zabezpečované zprávy
Detekované chyby lze vypsat jako jeden z výstupních vektorů.
V Matlabu je pro dekódování BCH kódů použit Berlekampův dekódovací algoritmus.

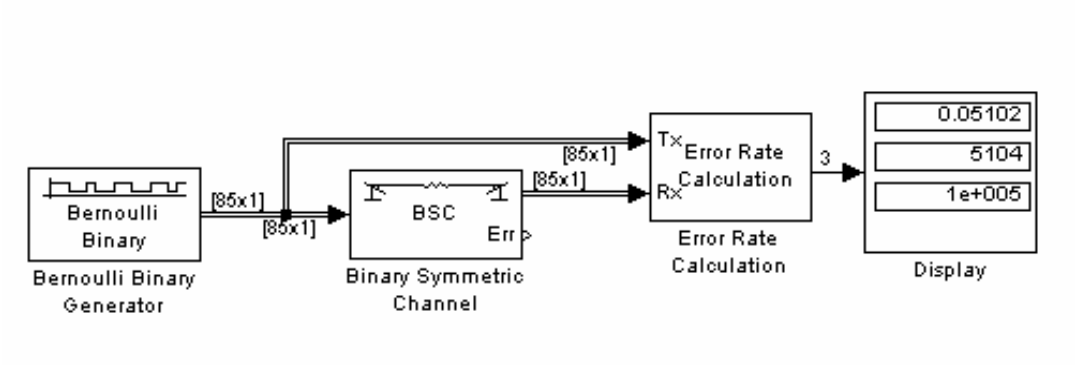


Obr. 3-1 Schéma BCH dekodéru v Matlabu



Obr. 3-2 Schéma BCH kodéru v Matlabu

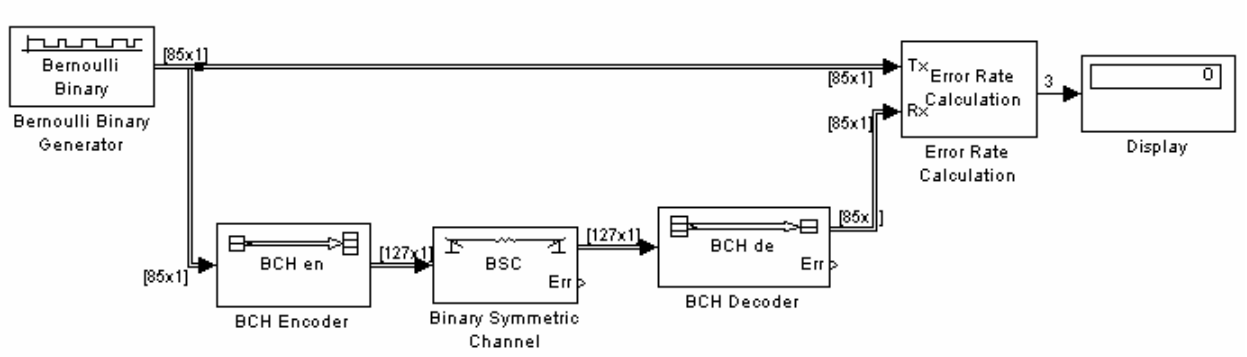
Základní zapojení přenosového systému bez zabezpečení BCH kódem



Obr. 3-3 Zapojení přenosového systému bez BCH kodeku

Zapojení s BCH kodekem

Výhodou tohoto zapojení je snadná změna parametrů přenosového kanálu a také možnost zvolit si BCH kodek a sledovat vliv zabezpečení na přenášená data.



Obr. 3-4 Zapojení přenosového systému s BCH kodekem

3.3 Výsledky simulace BCH kódů v Matlabu

Během testu bylo přenášeno 100 000 bitů, chybovost přenosového kanálu „Error Probability“ byla nastavena na 0,05. Z těchto hodnot vyplývá, že bez použití protichybového zabezpečení by mělo vzniknout po přenesení všech bitů přibližně 5 000 chyb. Do simulace jsem zařadil všechny BCH kódy do délky bloku *n = 127* bitů, které jsou schopny opravit *t = 1* až *15* chyb.

Tab. 3-1Výsledky Matlab

n	**K**	**T**	**Chyby**	**BER**	**R**	**ch**
[bit]	[bit]	[bit]	[bit]	[-]	[-]	[-]
7	4	1	1980	0,01980	0,57143	0,14286
15	11	1	3885	0,03885	0,73333	0,06667
15	7	2	1015	0,01015	0,46667	0,13333
15	5	3	162	0,00162	0,33333	0,20000
31	26	1	5499	0,05499	0,83871	0,03226
31	21	2	2853	0,02853	0,67742	0,06452
31	16	3	1046	0,01046	0,51613	0,09677
31	11	5	93	0,00093	0,35484	0,16129
31	6	7	5	0,00005	0,19355	0,22581
63	57	1	6094	0,06094	0,90476	0,01587
63	51	2	5071	0,05071	0,80952	0,03175
63	45	3	3429	0,03429	0,71429	0,04762
63	39	4	1895	0,01895	0,61905	0,06349
63	36	5	1005	0,01005	0,57143	0,07937
63	30	6	433	0,00433	0,47619	0,09524
63	24	7	187	0,00187	0,38095	0,11111
63	18	10	1	0,00001	0,28571	0,15873
63	16	11	0	0,00000	0,25397	0,17460
63	10	13	0	0,00000	0,15873	0,20635
63	7	15	0	0,00000	0,11111	0,23810
127	120	1	5807	0,05807	0,94488	0,00787
127	113	2	5745	0,05745	0,88976	0,01575
127	106	3	5215	0,05215	0,83465	0,02362
127	99	4	4700	0,04700	0,77953	0,03150
127	92	5	4023	0,04023	0,72441	0,03937
127	85	6	3069	0,03069	0,66929	0,04724

chyby – počet chyb za přenosovým kanálem zabezpečeným BCH kódem
BER – „bit error rate“ *BER* = počet chybných bitů/počet všech přenesených bitů

R – informační poměr kódu - $R = \frac{k}{n}$; (3.2)

ch - $ch = \frac{t}{n}$ (3.3)

4. Výběr a sestavení BCH kódu podle zadání

4.1 Výběr kvalitního BCH kódu

Jedním z hlavních kritérií při výběru správného bezpečnostního kódu je, aby počet *k* informačních znaků byl velký (znamená to, že bezpečnostní kód bude mít malou redundanci) a také to, aby minimální Hammingova vzdálenost *d* kódu byla velká (kód bude schopen objevovat a opravovat hodně chyb). Protože jsou tyto dva požadavky rozporné, hledáme vhodný kompromis. Tento parametr lze vyjádřit poměrem počtu informačních znaků *k* počtu všech znaku $R = \frac{k}{n}$ (4.1)

Parametr *R* určuje **informační poměr kódu**. Pomocí tohoto parametru lze také zjistit snížení přenosové rychlosti (čím vyšší hodnota *R*, tím je vyšší přenosová rychlost a kód má vyšší účinnost). Dalším významným parametrem je **zabezpečovací schopnost kódu** $ch = \frac{t}{n}$ (4.2) nebo

$ch = \frac{t}{n} \cdot 100$ [%] (4.3)

Požadavkem zadání je, aby výsledný BCH kód byl schopen opravit až *t = 6* nezávislých chyb v bitovém toku až *150* bitů. Vycházím z tabulky 3-1 uvedené v kapitole 3. Z této tabulky je zřejmé, že zadání splňují dva kódy BCH (63, 30) a kód BCH (127, 85).

Tab. 4-1 BCH kódy pro t = 6 chyb a n = 150 bitů

n	k	t	Chyby	BER	R	ch
[bit]	[bit]	[bit]	[bit]	[-]	[-]	[-]
63	30	6	433	0,00433	0,47619	0,09524
127	85	6	3069	0,03069	0,66929	0,04724

n – maximální délka kódového slova v bitech

k – počet informačních bitů v kódovém slově

t – maximální počet bitů, který může být opraven

R – informační poměr – podle vzorce 4.1

ch – zabezpečovací schopnost kódu – podle vzorce 4.2

Z tabulky 4-1 je zřejmé, že kód BCH (63, 30) má horší informační poměr *R* než kód BCH (127, 85), ale naopak má lepší zabezpečovací schopnost kódu *ch.*

Dalším pohledem na jednotlivé BCH kódy může být **složitost jejich realizace.** Funkce kodéru je pro všechny výše uvedené BCH kódy stejná a vychází z realizace obvyklé pro cyklické kódy. Tyto realizace vychází ze zapojení pomocí kruhového posuvného registru se zpětnými vazbami a zapojenými sčítačkami modulo 2.

Pro porovnání složitosti můžeme vzít počet potřebných paměťových buněk, které budou potřeba pro realizaci kodéru pomocí kruhového registru. Počet paměťových buněk získáme z řádu vytvářecího mnohočlenu kódu $r = n - k$. (4.4)

Pro kód BCH (63, 30) je tedy podle (4.4) potřeba 33 paměťových buněk a pro kód BCH (127, 85) je potřeba 42 paměťových buněk.

Kodeky se však mohou lišit i ve fázi dekódování (přesněji řečeno v metodě stanovení chybového mnohočlenu – lokátoru chyb), metody pro kontrolu správnosti a opravení chyb jsou stejné.

Podle informací uvedených výše jsem se rozhodl pro kód **BCH (63, 30)**.

4.2 Sestavení vytvářecího mnohočlenu *g(x)*

Kód BCH (63, 30) má následující parametry:

- $n = 2^m - 1,\ n = 63$ bitů maximální délka kódového slova v bitech
- $t = 6$ maximální počet chybných bitů, které mohou být opraveny
- $k \geq n - mt,\ k = 30$ *bitů* počet informačních bitů v kódovém slově
- $d_{min} \geq 2t + 1,\ d_{min} = 13$ minimální Hammingova vzdálenost

Nyní je potřeba získat vytvářecí mnohočlen *g(x)* tohoto kódu. Podle [6] se při hledání vytvářecího mnohočlenu vychází z Bóse – Chaudhuriho teorému.

1. Je zvolen primitivní mnohočlen stupně *m* a zkonstruované Galoisovo těleso GF (q^m),
2. Jsou nalezeny minimální mnohočleny $\mathbf{m}_j(x)$ pro α^j, kde $j = 1, \ldots, 2t$, a α je primitivní prvek Galoisova tělesa
3. Generující mnohočlen BCH kódu je $\mathbf{g}(x) = LCM\{\mathbf{m}_1(x), \mathbf{m}_2(x), \ldots, \mathbf{m}_{2t}(x)\}$.
   LCM (Least Common Multiple – nejmenší společný násobek)

Pro kód BCH (63, 30) platí:

1. Primitivní mnohočlen stupně *m = 6* je $p(X) = 1 + X + X^6$
2. Použijeme mnohočleny *i* = 1 až 6 z tabulky 4-2
3. Získáme vytvářecí mnohočlen **g***(x)*

Tab. 4-2 Nerozložitelné mnohočleny nad GF(2^6)

i	Nerozložitelný mnohočlen $\mathbf{m}_i$	Kořeny mnohočlenu
1	$1 + x + x^6$	$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}$
2	$1+x+x^2+x^4+x^6$	$\alpha^3, \alpha^6, \alpha 12, \alpha^{24}, \alpha^{48}, \alpha^{33}$
3	$1+x+x^2+x^5+x^6$	$\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{17}, \alpha^{34}$
4	$1+x^3+x^6$	$\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56}, \alpha^{49}, \alpha^{35}$
5	$1+x^2+x^3$	$\alpha^9, \alpha^{18}, \alpha^{36}$
6	$1+x^2+x^3+x^5+x^6$	$\alpha^{11}, \alpha^{22}, \alpha^{44}, \alpha^{25}, \alpha^{50}, \alpha^{37}$

Kompletní tabulka je uvedena např. v literatuře [6].

Vytvářecí mnohočlen kódu BCH (63, 30):
$\mathbf{g}(x) = (1+x+x^6).(1+x+x^2+x^4+x^6).(1+x+x^2+x^5+x^6).(1+x^3+x^6).(1+x^2+x^3).(1+x^2+x^3+x^5+x^6)$

$$= \mathbf{1+x+x^2+x^5+x^6+x^8+x^9+x^{11}+x^{13}+x^{14}+x^{15}+x^{20}+x^{22}+x^{23}+x^{26}+x^{27}+x^{28}+x^{29}+x^{30}+x^{32}+x^{33}} \quad (4.5)$$

Vytvářecí mnohočlen BCH kódu lze např. získat pomocí programu Matlab a jeho funkce „bchgenpoly“. Pomocí této funkce lze získat vytvářecí mnohočlen pro *m = 3* až *16*.

Př.:
g(x) = bchgenpoly(63,30)

Funkce bchgenpoly vrací stejný polynom, který byl získán pomocí literatury [6].

5. Kodér BCH kódu

Při konstrukci kodéru binárního BCH kódu je možné postupovat jako při konstrukci kodéru cyklického kódu [1], [5].

Máme:

- **m**(x) - mnohočlen bloku nezabezpečené zprávy
- **g**(x) - vytvářecí mnohočlen kódu
- **q**(x) - mnohočlen podílu
- **r**(x) - mnohočlen zbytku
- **c**(x) - mnohočlen zabezpečené zprávy

BCH kód je vyjádřen pomocí vytvářecího (generujícího) mnohočlenu **g**(x). Řád tohoto mnohočlenu je *r = (n – k).*

Postup kódování můžeme rozdělit do tří kroků:

1. Vynásobíme mnohočlen nezabezpečené zprávy **m**(x) členem $x^{(n-k)}$.

2. Následně je tento mnohočlen vydělen vytvářecím mnohočlenem kódu **g**(x). Po dělení těchto dvou mnohočlenů získáme mnohočlen zbytku **r**(x).

3. Tento mnohočlen zbytku **r**(x) je přičten k součinu **m**(x) * $x^{(n-k)}$ a tím získáme zabezpečenou zprávu **c**(x) [5].

Pro kódování podle výše uvedeného postupu můžeme použít vztah

$$\frac{\mathbf{m}(x).x^{(n-k)}}{\mathbf{g}(x)} = \mathbf{q}(x) + \frac{\mathbf{r}(x)}{\mathbf{g}(x)} \qquad (5.1)$$

který upravíme na

$$\mathbf{m}(x).x^{(n-k)} = \mathbf{q}(x).\mathbf{g}(x) + \mathbf{r}(x)\,. \qquad (5.2)$$

Podle 3. bodu můžeme zapsat

$$\mathbf{m}(x).x^{(n-k)} + \mathbf{r}(x) = \mathbf{q}(x).\mathbf{g}(x) + \mathbf{r}'(x)\,. \qquad (5.3)$$

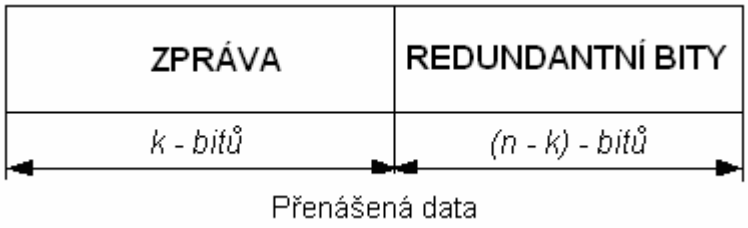
Pokud během přenosu nedojde k chybám $\mathbf{r}(x) = \mathbf{r}'(x)$

můžeme napsat

$$\mathbf{c}(x) = \mathbf{q}(x).\mathbf{g}(x)\,. \qquad (5.4)$$

Ze vztahu 5.4 je vidět, že pokud během přenosu nedojde k chybám je mnohočlen zabezpečené zprávy bezezbytku dělitelný vytvářecím mnohočlenem kódu.

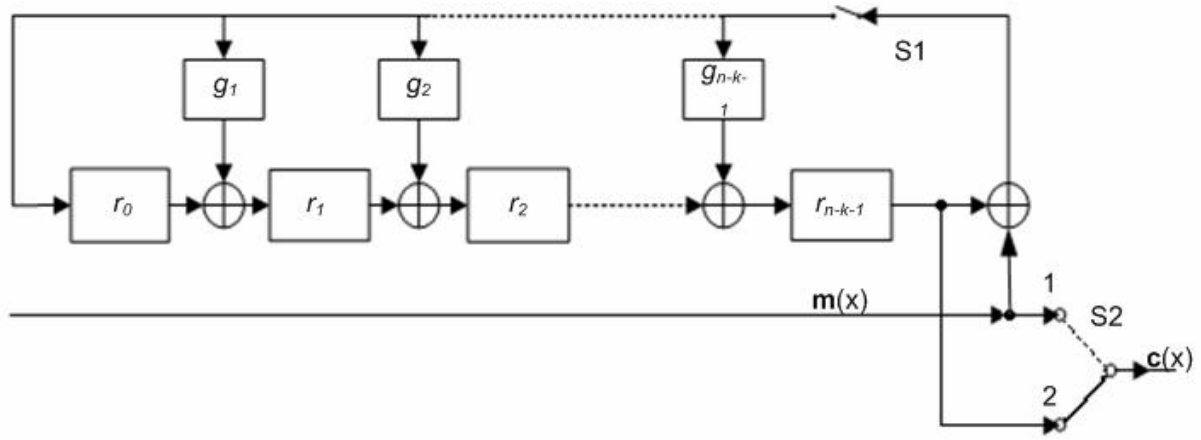
BCH kódy patří mezi systematické kódy, proto je kódové slovo rozděleno do struktury, která je na obrázku 5.1.



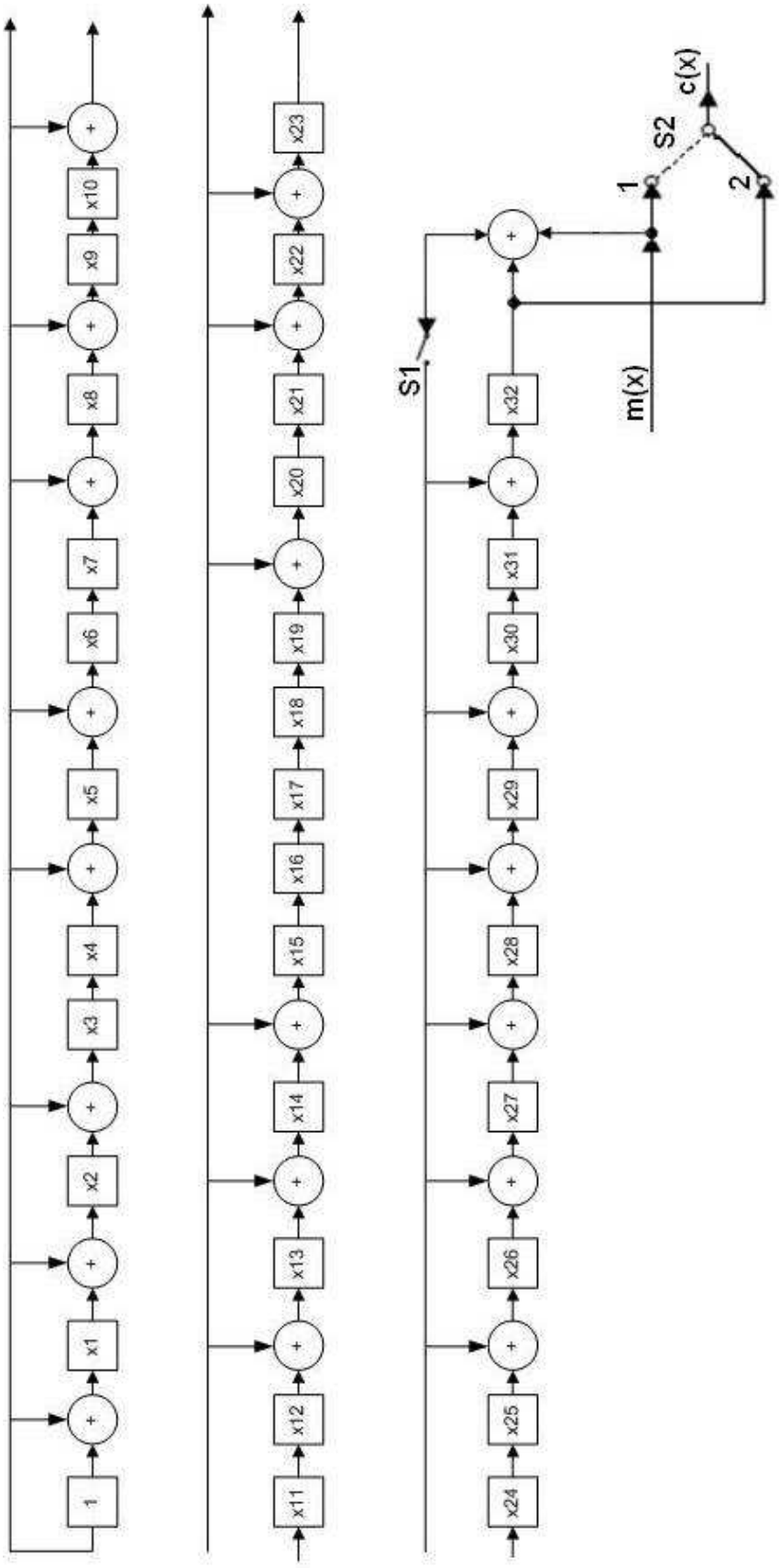
Obr. 5-1 Rozložení bitů v kódovém slově

5.1 Realizace pomocí posuvných registrů

Postup uvedený výše lze realizovat pomocí lineárního kruhového posuvného registru, který má zapojeny zpětné vazby a sčítačky modulo 2 podle vytvářecího mnohočlenu vybraného kódu. V některé literatuře se uvádí zkratka LFSR – Linear Feedback Shift Register [6].



Obr. 5-2 Posuvný kruhový registr



Obr. 5-3 Kodér kódu BCH (63,30) pomocí kruhového posuvného registru

Zapojení kodéru:

- Posuvný registr obsahuje (*n* – *k*) paměťových buněk. Počet těchto paměťových buněk potřebných pro kodér můžeme zjistit podle řádu vytvářecího mnohočlenu **g**(x),

- Z vytvářecího mnohočlenu **g**(x) zjistíme, před kterými členy je sčítací znaménko a před stejné členy v posuvném registru zařadíme sčítačky modulo 2. Nejvyšší řád vytvářecího mnohočlenu již nemá svoji buňku v posuvném registru a sčítačka, která mu náleží, je umístěna před buňkou X_0.

Popis funkce:

- Nejprve do kodéru vstupuje *k* informačních bitů. Spínač S2 je v poloze 1 takže tyto bity také z kodéru vycházejí v nezměněné podobě. Spínač S1 je po dobu přijímání informačních bitů sepnut. Díky sepnutému spínači S1 se dostávají informační bity do posuvných registrů. Tento děj trvá od 1 do *k* cyklů,

- V cyklech od *k* + 1 do *n* je vyprazdňován posuvný registr a všechna data jsou přenášena na výstup kodéru. Spínač S1 je rozepnut a přepínač S2 je v poloze 2,

- Celkový počet posunutí v posuvných registrech je *n*. Po provedení tohoto postupu je na výstupu připraveno kódové slovo (zabezpečená zpráva) a proces se opakuje s další bitovou posloupností. Výsledné kódové slovo má strukturu stejnou, jaká je uvedena na obrázku 5-1.

6. Dekodér BCH kódu

V této kapitole budou rozebrány metody pro dekódování BCH kódu. Proces dekódování BCH kódu je oproti kódování výpočetně náročnější. Účelem dekódování je z přijaté zprávy, ve které došlo vlivem rušení ke změnám, vytvořit zprávu shodnou s vyslanou zprávou, za předpokladu, že nebyla překročena zabezpečovací schopnost kódu. BCH dekodér pro kód s plánovanou vzdáleností ($2t + 1$) je schopen opravit až t – násobné chyby.

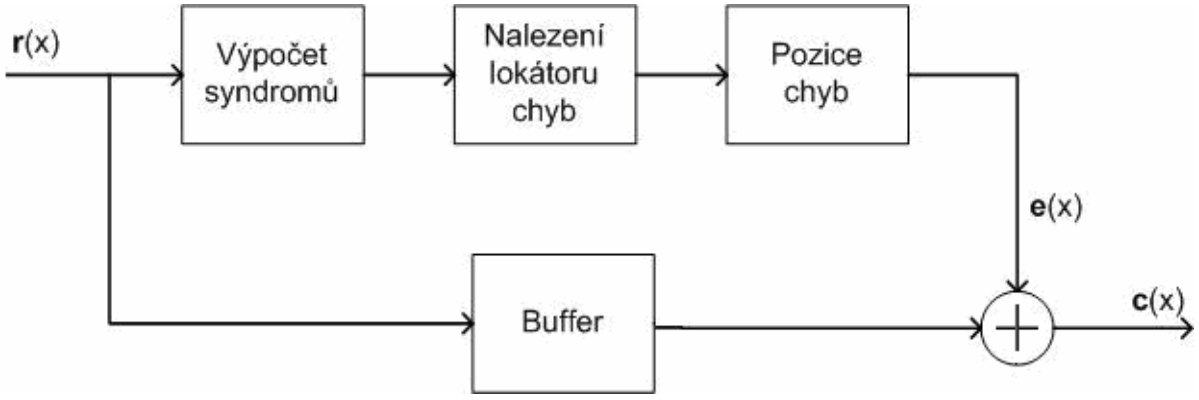
Mějme tedy vyslané kódové slovo **c**(x), které je ovlivněno během přenosu rušením. Přijmeme tedy pozměněné kódové slovo **r**(x). Potom podle [6] můžeme napsat vztah:

$$\mathbf{r}(x) = \mathbf{c}(x) + \mathbf{e}(x), \tag{6.1}$$

kde **e**(x) je chybový mnohočlen. Pokud tedy existuje přijatý mnohočlen **r**(x), který obsahuje t nebo méně chyb, snažíme se najít chybový mnohočlen **e**(x). Pokud však přijatý mnohočlen obsahuje více jak t chyb je překročena zabezpečovací schopnost kódu a dekódovaní skončí neúspěšně.

Dekódovací proces můžeme rozdělit na tři fáze:

1. Kontrola správnosti přijatého kódového slova – získání syndromových rovnic
2. Určení chybového mnohočlenu – lokátoru chyb
3. Nalezení kořenů chybového mnohočlenu (lokátoru chyb) a opravení chyb [6]



Obr. 6-1 Schéma dekódování

6.1 Kontrola správnosti přijatého kódového slova

Vycházíme ze vztahu 6.1. [6], kdy přijaté kódové slovo můžeme zapsat ve tvaru:

$$\mathbf{r}(x) = r_0 + r_1 x + ... + r_{n-1} x^{n-1} \qquad (6.2)$$

V prvním kroce musíme zjistit, zda vůbec k nějakým chybám během přenosu došlo. To můžeme zjistit tak, že do mnohočlenu přijatého kódového slova dosadíme postupně kořeny minimálních mnohočlenů, ze kterých byl sestaven vytvářecí mnohočlen **g**(x).

$$r(\alpha) = r(\alpha) = r(\alpha^2) = ... = r(\alpha^{2t}) = 0 \qquad (6.3)$$

Pokud je přijaté kódové slovo bez chyb, jsou všechny syndromové rovnice nulové, v opačném případě získáme soustavu rovnic, kterou musíme vyřešit. Pro BCH kód opravující *t*-chyb bude těchto syndromů 2*t*.

$$S = (S_1, S_2, \ldots, S_{2t}) \qquad (6.4)$$

Získání syndromů z přijatého mnohočlenu:

1. Vydělíme přijatý mnohočlen **r**(x) minimálním mnohočlenem $\mathbf{\Phi_i}(x)$
2. Získáme rovnici $\mathbf{r}(x) = \mathbf{a}(x)\,\mathbf{\Phi_i}(x) + \mathbf{b}(x)$ (6.5)
3. poté můžeme napsat, že syndrom se rovná $S_i = b(\alpha^i)$

a(x) je mnohočlen podílu

b(x) je mnohočlen zbytku

Syndrom $S_i = b(\alpha^i)$ lze najít pomocí lineárního posuvného registru se zpětnými vazbami zapojenými podle minimálního mnohočlenu $\mathbf{\Phi_i}(x)$.

6.2 Určení chybového mnohočlenu – lokátoru chyb

Ve 2. kroku dekódování binárního BCH kódu hledáme řešení soustavy syndromových rovnic a výsledný chybový mnohočlen, jehož kořeny nám určí, kde nastaly chyby – lokátor chyb [6]. Tento krok dekódování je nejsložitější částí, která je velice náročná na výpočetní výkon. Existuje několik rozdílných postupů jak tento krok vyřešit. V další části budou uvedeny dva algoritmy, které se dnes ve většině případů používají a to algoritmus Petersonův a Berlekamp – Massey.

V tomto kroku je nutné sestavit chybový mnohočlen ve tvaru [6]:

$$\mathbf{k}(x) = (1 - k_1 x)(1 - k_2 x)...(1 - k_p x) \qquad (6.6)$$

kde *p* je počet chyb, který nastal v přijatém kódovém slově.

Rovnici 6.6 můžeme podle [6] zapsat ve tvaru:

$$\mathbf{k}(x) = k_0 + k_1 x + k_2 x^2 + \ldots + k_p x^p \qquad (6.7)$$

koeficient $k_0 = 1$, ostatní koeficienty je nutné dopočítat.

Petersonův algoritmus pro určení chybového mnohočlenu

Vztah mezi syndromy a prvky chybového mnohočlenu můžeme vyjádřit pomocí součinu matic:

$$\mathbf{M} \times \mathbf{K} = -\mathbf{D} \qquad (6.8)$$

kde **M**, **K** a **D** jsou postupně:

$$\begin{bmatrix} S_l & S_{l-1} & \ldots & S_1 \\ S_{l+1} & S_l & \ldots & S_2 \\ \ldots & \ldots & \ldots & \ldots \\ S_{2l-1} & S_{2l-2} & \ldots & S_l \end{bmatrix} \times \begin{bmatrix} k_1 \\ k_2 \\ \ldots \\ k_l \end{bmatrix} = -\begin{bmatrix} S_{l+1} \\ S_{l+2} \\ \ldots \\ S_{2l} \end{bmatrix} \qquad (6.9)$$

kde *t* je počet chyb, který v přijaté zprávě nastal.

Postup:

1. Nejprve do proměnné *l* přiřadíme maximální počet chyb, který může vybraný BCH kód opravit *l* = *t.*
2. Sestavíme matice podle 6.9.
3. Vypočítáme determinant matice syndromů *det*(**M**), pokud je tento determinant roven 0 (*det*(**M**) = 0) snížíme hodnotu *l* = *l* – 1 a vrátíme se ke kroku 2. Pokud však determinant není roven 0 (*det*(**M**) ≠ 0), je právě hodnota *l* rovna počtu chyb, který nastal.
4. Dále se z matic dopočítají koeficienty lokátoru chyb k_l ($0 < l \leq t$) a dosadí se do rovnice 6.7.

Berlekamp – Massey (BMA) algoritmus pro určení chybového mnohočlenu

Tato metoda využívá efektivní iterativní techniku pro nalezení chybového mnohočlenu – lokátoru chyb. Výhodné je, že není nutné předem znát počet chyb [14].

V tomto algoritmu je chybový mnohočlen **k**(x) získán pomocí *t* - 1 iterací.

V průběhu každé iterace *r* je řád chybového mnohočlenu inkrementován o 1. Řád chybového mnohočlenu tedy určuje počet chyb, který v přenášené zprávě nastal a kořeny tohoto chybového mnohočlenu jsou spojeny s chybami.

BMA vychází z toho, že pro počet iterací *r* větší nebo roven počtu chyb *l,* který ve zprávě nastal ($r \geq l$) je odchylka d_r (v cizojazyčné literatuře discrepancy) rovna 0 ($d_r = 0$).

$$d_r = \sum_{j=0}^{t} S_{2r-j+1} \cdot k_j \qquad (6.10)$$

Pokud však je ($r < l$) odchylka vypočítaná podle vzorce 6.10 není nulová a je nutné pokračovat v dalším kroku zvýšením řádu a přepočítáním koeficientů chybového mnohočlenu.

Pokud je počet iterací *r* vetší než počet chyb *t,* který je kód schopen opravit je v přijaté zprávě více jak *t* chyb a dekódování skončí neúspěšně [6].

6.3 Určení pozice chyb a jejich oprava

Určení pozice chyb a jejich oprava je posledním krokem dekódování, v tomto kroce již známe chybový mnohočlen – lokátor chyb **k**(x). Většinou se tento krok provádí metodou hrubé síly v některé literatuře se tento postup nazývá Chien search, který spočívá v postupném dosazování všech prvků z Galoisova tělesa (kromě nuly), které používáme. Postupným dosazováním získáme kořeny, pro které platí $\mathbf{k}(\alpha^i) = 0$. Po získání kořenu chybového mnohočlenu opravíme právě *i*-té místo [6].

7. Možné realizace BCH kodeku

V předchozích kapitolách jsem se věnoval především teoretickému popisu BCH kódů, možnostem a výběru kódu, který splňuje požadavky zadání. V této kapitole bych chtěl nejprve obecně rozvést možnosti realizace kodéru a dekodéru BCH kódu a následně se již zaměřit na sestavení kritérií pro výběr nejvhodnější varianty realizace a na návrh této realizace.

7.1 Soubor realizací BCH kodeku

Základní možnosti realizace kodeku můžeme zjednodušeně rozdělit na dvě:

- **Hardwarová realizace**

- **Softwarová realizace**

Obě metody mají určité výhody i nevýhody, které se v následujícím textu budu snažit přiblížit. Pro správný výběr způsobu realizace je také vhodné znát, pro jaký účel bude kodek použit a jaký typ přenosu bude kód zabezpečovat – druh přenosového kanálu. V závislosti na těchto informacích může být výhodnější realizace hardwarová např. z důvodů rozměrů nebo nutnosti integrovat tento protichybový systém do komplexnějšího hardwarového celku.

Pod hardwarovou realizací si však v dnešní době nemůžeme představit zapojování diskrétních elektrických součástek, ale návrh ve specializovaném hardwarově orientovaném jazyce HDL (Hardware Description Language) a následnou simulaci a překlad funkčního návrhu do specializovaného hardwaru.

Softwarová realizace může být naopak výhodná pro snadnou realizaci bez nutnosti dalšího speciálního hardwaru. Další výhodou může být snadná modifikace kódu a testování.

V zadání práce nebylo přesně určeno, která realizace by měla být upřednostněna nebo by byla vhodnější vzhledem k dalšímu uplatnění kodeku.

Možnosti hardwarová realizace

1. Zapojení tvořené číslicovými integrovanými obvody
2. Mikrokontrolery
3. Programovatelnými logickými obvody – PLD, CPLD a FPGA

Zapojení tvořené číslicovými integrovanými obvody

Řešení pomocí číslicových obvodů základních řad se již v dnešní době moc nepoužívá, realizace tímto způsobem byly rozšířeny v 70. letech 20. století. Pomocí těchto obvodů lze realizovat malý rozsah jednodušších pomocných funkcí. Ve většině případů je výhodnější pro realizaci použít jinou formu dovolující větší integraci. Pro odzkoušení a realizaci BCH kodeku by byla tato metoda velice složitá a neefektivní.

Výhody:

- rychlá reakce
- malá spotřeba – pro obvody CMOS
- snadná dostupnost

Nevýhody:

- větší počty pouzder – omezená možnost miniaturizace – malý stupeň integrace
- komplikace při změně funkce, nutnost změny plošného spoje při úpravě zapojení – není vhodné pro testování

Tato metoda je asi ze všech nejméně vhodná pro realizaci BCH kodeku.

Zapojení s mikrokontroléry

Mikrokontroléry (mikropočítače) umožňují po vhodném naprogramování řešit nejrůznější úlohy. Výhodou je také možnost přizpůsobení navrhovaného subsystému požadavkům pomocí programu, např: návrh zrealizovat v jazyce C.

Mikrokontroléry, můžeme se setkat i s označením jednočipové mikropočítače, obsahují v jednom pouzdře několik základních částí:

- řadič s aritmetickologickou jednotkou. U mikrokontrolérů je obvyklá délka slova 4, 8, 16, nebo 32 bitů

- paměť programu, tato paměť může být typu EPROM nebo Flash u programovatelných mikrokontrolérů. Pro aplikace pro sériovou výrobu se většinou používá paměť typu ROM.
- paměť dat typu read/write (Harwardská architektura)
- další periferní obvody pro vstup a výstup dat

Výhody realizace pomocí mikropočítače a mikrokontroleru:

- možnost změnit vykonávanou funkci přeprogramováním, bez nutnosti měnit plošný spoj
- možnost využít rozmanité parametry – různé rychlosti a velikosti
- lze realizovat složité algoritmy
- univerzálnost

Nevýhody této realizace

- nižší rychlost reakce
- složitost
- cena

Pro realizaci BCH kodeku by bylo možné použít např. mikrokontroléry a vývojové kity od firmy Atmel. Např. mikrokontrolér ATmega32 a dále by bylo potřeba použít vývojový kit Atmel STK500 obsahující programátor (programovací deska) ovládací a programovací software.
Programátor zprostředkovává komunikaci mezi počítačem a mikrokontrolérem. K počítači se většinou připojuje přes paralelní nebo sériový port u modernější i přes USB (Universal Serial Bus).

Zapojení s programovatelnými logickými obvody

Programovatelné logické obvody se v dnešní době začínají stále více prosazovat a s pohledem do budoucnosti je tento způsob velice perspektivní. V dnešní době již poskytují návrhářům dostatečný výkon a rychlost prováděných úkonů. Podle [11] použití programovatelných logických obvodů snižuje celkové náklady potřebné na vývoj a výrobu obvodů podle potřeb zákazníka. Výhodou je možnost přeprogramování podle potřeb (velmi výhodné pro testování aplikace) pomocí vývojového prostředí,

které většinou dodává výrobce programovatelného obvodu. Dnešní FPGA (Field Programmable Gate Arrays) poskytují miliony hradel pracujících na rychlosti 300 MHz a tyto obvody jsou vyráběny 0,13-mikronovou technologií (Xilinx). Cena těchto obvodu se pohybuje kolem 10 $ za kus [11].

Podle [9] se v průběhu vývoje ustálily dvě hlavní skupiny těchto obvodů:

- obvody s „hrubozrnnou“ strukturou základních bloků (makrobuněk), které jsou určeny pro jednoprůchodové realizace logických funkcí
- obvody s „jemnozrnnou“ strukturou, u těchto obvodů se počítá s propojením základních bloků

Jednoprůchodová realizace znamená, že funkce je realizována jednou makrobuňkou, bez nutnosti propojovat větší počet makrobuněk pro realizaci této funkce. Výstup může být použit pro vstup do další buňky [9].

Rozdělení logických programovatelných obvodů:

- **klasické PLD** (Programmable Logic Devices)
- **CPLD** (Complex programmable logic device)
- **FPGA** (Field Programmable Gate Arrays)

Klasické programovatelné logické obvody PLD (Programmable Logic Devices) jsou podobným řešením jako s logickými obvody základních řad, ale obvody PLD jsou realizovány pouze jediným obvodem. Výhodou je, že jejich funkce se dá jednoduše změnit přeprogramováním. Rychlost reakce těchto obvodů je velice dobrá.
Jedná se o jednoduché programovatelné logické pole AND/OR. První programovatelné logické obvody se začínají vyvíjet v druhé polovině 70. let.
PLD obvody patří do skupiny s „hrubozrnnou“ strukturou makrobuněk a jsou vhodné pro jednodušší až středně složité aplikace.
Základem makrobuněk je dvoustupňová struktura používaná při realizaci kombinačních logických obvodů zapsaných ve tvaru součtu součinů [9].

CPLD (Complex Programmable Logic Device)
jsou určitým mezistupněm mezi obvody PLD a FPGA.

FPGA (Field Programmable Gate Arrays)

Pomocí obvodů FPGA lze dnes vytvořit procesory softwarově. Realizace pomocí FPGA obvodu je vhodná tam, kde nedostačuje rychlost reakce mikropočítače. Obvody architektury FPGA jsou založeny na malých generátorech logických funkcí s paměťmi, klopných obvodech a mnoha horizontálních a vertikálních propojení. Obvody FPGA mají nejobecnější strukturu, a proto umožňují realizaci nejsložitějších číslicových systémů [15].

Nejznámější výrobci jsou firmy Xilinx a Altera.

Možnosti a postup při programování logických obvodů

Programování logických obvodů je možno provádět pomocí:

- samostatných programátorů řízených počítačem – tato metoda se používá u jednodušších obvodů
- přímo v systému – In System Programming – dnes rozšířená metoda u obvodů FPGA [8]

Postup při vývoji aplikací pro programovatelné logické obvody:

- zápis vstupních údajů
- funkční simulace zdrojového textu
- syntéza, implementace
- simulace modelu vzniklého implementací
- naprogramování cílového obvodu, odzkoušení [9]

Návrhové systémy:

Lze použít návrhové systémy od výrobců programovatelných obvodů nebo univerzální systémy, které jsou však ve většině případů drahé.

Programování logických obvodů se provádí ze vstupních údajů, které jsou zapsány v některém HDL jazyce (VHDL, Verilog) nebo v grafické formě (schéma, diagram) převedením syntézou na netlist, který se pak implementuje do zvoleného cílového obvodu [16].

Výhody:

- rychlá reakce
- snadná modifikace – návrh ve HDL jazyce, možnost přeprogramovat zapojený obvod
- univerzálnost
- vývojové prostředí lze od výrobce získat zdarma

Softwarová realizace

Principem softwarové realizace je naprogramování kodéru a dekodéru BCH kódu, který byl vybrán pomocí kritérií a splňuje podmínky zadání ve vyšším programovacím jazyce např. C nebo C++ a požití např. pro zabezpečení ukládaných dat nebo pro komunikaci přes sériový port.

Výhody:

- jednoduchá modifikace programu v průběhu testování

Nevýhody:

- rozměry – realizace na vyhrazeném PC

7.2 Výběr realizace - kritéria

BCH kódy využívají složité algoritmy a výběr jejich realizace není snadný. Při výběru realizace záleží i na tom, kde má být protichybový kodek nasazen a tím mohou být kladeny požadavky na jeho rozměry, rychlost operací atd.. V této kapitole budou sestaveny kritéria pro kodér a dekodér BCH kódu. Pro zjednodušení jsou všechna kritéria považována za stejně významná.

- Kritéria – kodér

V kapitole 3 byly porovnány jednotlivé BCH kódy až do délky n = 127. Z tabulky 3-1 je zřejmé, že zadání splňují kódy BCH (63,30) a BCH (127,85). Oba tyto kódy jsou schopné opravit až t = 6 nezávislých chyb. Budou porovnávány dva BCH kódy, které opravují stejný počet chyb. Za hlavní srovnávací kritéria bude považován informační poměr kódu R, konstrukční složitost S.

Informační poměr BCH kódu je dán vztahem $R = k/n$ (7.1)

Konstrukční složitost kodéru BCH kódu je především dána potřebným počtem paměťových buněk logických operátorů. Potřebný počet těchto buněk můžeme určit podle řádu vytvářecího mnohočlenu **g**(x). Řád vytvářecího mnohočlenu je dán vztahem

$$S = n - k \qquad (7.2)$$

$R_{63} = 30/63 = 0{,}48$

$R_{127} = 85/127 = 0{,}66$

$S_{63} = 33$ paměťových buněk

$S_{127} = 42$ paměťových buněk

Ze zjištěných hodnot je vidět, že kód BCH (127,85) má lepší informační poměr, ale větší konstrukční složitost. Obecně lze říci, že kód BCH (63,30) je vhodnější k realizaci, protože dokáže opravit požadovaný počet chyb v kratší bitové posloupnosti *n* a má lepší konstrukční složitost – to se projeví hlavně ve fázi dekódování.

V tabulce 7-1 jsou uvedeny základní kritéria pro výběr realizace. Jednotlivým realizacím jsou přiřazeny známky od 1 do 3 (1 = nejlepší). Realizace jsou porovnávány podle výsledných rozměru („Rozměry“), složitosti realizace a požadavků kladených na znalosti při realizování danou metodou („Složitost“), podle možnosti snadno proveditelných úprav již realizovaného kodeku („Modifikovatelnost“), podle integrace do PKS – protichybového kódového systému a také podle ceny realizace („Cena“).

Tab. 7-1 Kritéria pro výběr realizace

Realizace	Rozměry	Složitost	Modifikovatelnost	Integrace do PKS	Cena
číslicové integrované obvody	3	3	3	3	3
mikrokontrolery	2	2	2	2	2
Logické obvody – FPGA	1	2	1	1	2
soft. PC	3	2	1	3	2

Podle tabulky je nejvhodnější realizace pomocí logických programovatelných obvodů a to zvláště obvody FPGA. Jedná se o moderní možnost realizace, která se v poslední době začíná stále častěji prosazovat. U této metody se již nedá přímo hovořit o hardwarové realizaci, protože návrh obvodu probíhá ve speciálním programovacím jazyce např. VHDL a následném naprogramování obvodu. Tato realizace přináší mnohé výhody jako např. možnost přeprogramování již zapojeného obvodu. Tato realizace mi připadá jako nejvhodnější.

8. Návrh realizace protichybového kodeku

V této kapitole bude uveden návrh realizace vybraného BCH kódu z předchozích kapitol. Nejprve si uvedeme prostředky, pomocí kterých je tento návrh možný, jednotlivé HDL jazyky a software, se kterým je možný překlad funkčního návrhu do hardwaru. Dále bude následovat návrh kodéru a dekodéru ve VHDL jazyce.

8.1 Prostředky a postupy pro programování FPGA obvodu

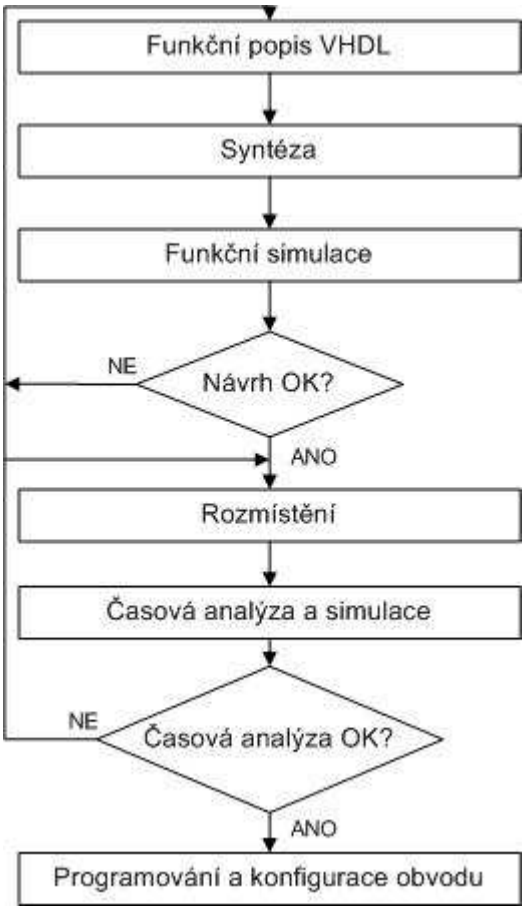
Popis složitějších číslicových obvodů se většinou provádí v některém HDL (Hardware Description Language) jazyce. V současnosti se nejvíce používají jazyky Verilog a VHDL (v dalším textu se zaměřuji pouze na VHDL). Vývoj jazyka VHDL byl zahájen již v roce 1981 v rámci výzkumného projektu, který měl zefektivnit vývoj velmi rozsáhlých integrovaných obvodů. Jazyky HDL umožňují popsat chování obvodu na různých úrovních abstrakce. Optimalizace návrhového systému na úrovni hradel, klopných obvodů je v těchto jazycích již hodně skryta a ponechána na kompilátoru a syntezátoru těchto jazyků.

Na trhu existují dva hlavní výrobci, kteří nabízí ucelenou řadu FPGA obvodů firma Xilinx a Altera. Oba výrobci poskytují ke svým obvodům i integrované vývojové prostředí obsahující kompletní softwarové nástroje pro návrh (např. Xilinx ISE WebPack nebo Altera Quartus II WebPack). Návrh však nemusí probíhat pouze pomocí těchto nástrojů, ale může být vytvořen v jazyce C v některém rozšířeném prostředí např. Visual Studio a poté návrh přeložit pomocí nástroje CoDeveloper do některého HDL jazyka.

V průběhu návrhu jsem používal vývojové prostředí Altera Quartus II Web Edition verze 7.2, které lze zdarma získat na webovských stránkách firmy Altera. Tato verze vývojového prostředí je bohužel omezena pouze na návrh obvodů z řad CPLD a jednodušších obvodů FPGA, ale to pro tento případ postačuje. Toto vývojové prostředí však obsahuje všechny potřebné softwarové nástroje pro kompletní návrh obvodu a testování (např. Chip planner – umožňuje zobrazit výsledné rozmístění a propojení jednotlivých bloků obvodu, Timing Analyzer – umožňuje ověřit výkonnost návrhu a požadavky na časování obvodu). Lze provést funkční (ověřuje se pouze správnost výstupů) i časovou (kontroluje se i dodržení časových parametrů) simulaci návrhu.

Proces návrhu obvodu FPGA

V této časti vycházím z informací uvedených v literatuře [16], [17] a z nápovědy, která je součástí vývojového prostředí Quartus II. Na následujícím diagramu je znázorněn postup při návrhu obvodu v prostředí Quartus II WebPack.



Obr. 8-1 Proces návrhu FPGA obvodu

Popis jednotlivých kroků:

1. Funkční popis – návrh obvodu pomocí některého HDL jazyka nebo schematického diagramu,
2. Syntéza – v tomto kroku se vygeneruje logické schéma ve formě netlistu, který je tvořen jednotlivými logickými bloky (LE – logic element),
3. Funkční simulace – po syntéze je provedena funkční simulace návrhu, při kterém se však nekontroluje dodržení časových parametrů,
4. Rozmístění (Fitting) – provádí se mapování a umístění logických bloků netlistu do struktury již konkrétního obvodu a následně se provede propojení jednotlivých bloků (routing),

5. Časová analýza a simulace – provádí se časová analýza obvodu (zpoždění průchodu signálu, atd.), a také se provádí funkční simulace, ve které se již sleduje dodržení časových parametrů obvodu,
6. Programování a konfigurace – v tomto kroku se provádí ladění na reálném obvodu. Po úspěšném odzkoušení na reálném obvodu se již tento obvod může dát do sériové výroby.

8.2 Návrh kodéru

Při návrhu kodéru BCH kódu se vychází z teorie uvedené v kapitole 5 a podle výběru kódu v následujících kapitolách – BCH (63,30) t = 6. Nyní bude popsán návrh kodéru pomocí jazyka VHDL. Při návrhu kodeku je nutné si uvědomit, že bude součástí kompletního protichybového systému, který se skládá i z dalších obvodů (generátor hodinových impulzů, obvody pro synchronizaci a řízení), které jsou nutné pro správnou funkci tohoto systému jako celku.

Popis obvodu pomocí jazyka VHDL je tvořen zápisem hlavních návrhových jednotek. Jednou z hlavních (povinných) návrhových jednotek jsou tzv. entity [15]. Entita slouží k popisu rozhraní (vstupy a výstupy) objektu, tento objekt může být značně rozsáhlý. Entitu si můžeme představit jako schematickou značku, která definuje parametry vstupů a výstupů obvodu (typ dat, směr přenosu dat). Entita nepopisuje funkci obvodu.

Příklad hlavní entity BCH kodéru:

```
ENTITY encoder_bch IS
PORT (clk, reset, din: IN BIT;
      vdin, dout: OUT BIT);
END encoder_bch;
```

Z uvedeného kódu je vidět, že název entity je „encoder_bch“ a v této entitě jsou definovány tři vstupní porty a dva výstupní porty – tyto vstupy / výstupy budou sloužit pro správnou komunikaci s okolím.

BCH kodér se skládá z dalších dvou entit, které se nazývají „ering“ – posuvný registr a „ecount“ – řídící systém. Řídící systém je složen z čítače a klopného obvodu (flip-flop). Tento obvod řídí „spínače“ S1 a S2 (viz teoretický popis kodéru) a je závislý na hodnotě čítače a stavu reset.

Chování entit (vlastní funkce) je definováno pomocí návrhové jednotky „architektura“. Každá entita musí mít alespoň jednu architekturu. Architektura popisuje chování, tok dat a strukturu každé entity.

Příklad začátku architektury:

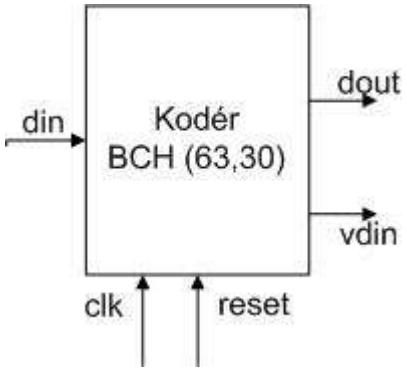
ARCHITECTURE enca OF encoder_bch IS

SIGNAL vdin1, rin, rout, rll: BIT;

Veškeré soubory kodéru a dekodéru jsou uloženy na přiloženém CD.

Popis vstupů a výstupů BCH kodéru:

- Reset – vstup - reset kodéru
- Din – vstup – vstup dat (nezabezpečená posloupnost bitů)
- Dout – výstup – výstup zabezpečených dat BCH kódem
- Clk – vstup – vstup hodinových impulzů
- Vdin – výstup – signál pro řízení vstupu, pokud je výstup vdin = log1 jsou načítána vstupní data a posílána na výstup dout (k – informační bity), pokud je vdin = 0 nejsou přijímány žádné data na portu din a na výstup dout jsou generovány zabezpečovací bity.



Obr. 8-2 Kodér BCH (63, 30)

8.3 Návrh dekodéru

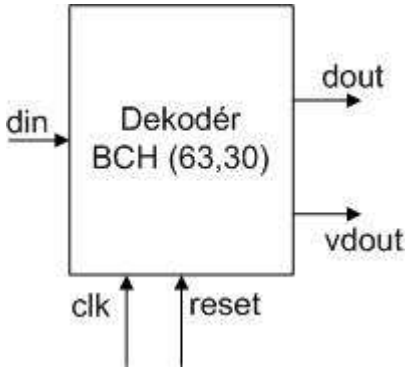
Návrh dekodéru je stejně jako kodér navržen ve VHDL jazyce a určen pro realizaci pomocí FPGA obvodu. V návrhu je opět pamatováno na nutnost integrace tohoto kodeku do protichybového kódového systému (vstupy clk, reset).

Dekódování přijatého kódového slova je rozděleno do tří částí (viz. kapitola 6 - Dekodér BCH kódu). Dekodér obsahuje následující entity, které slouží pro:

- Výpočet syndromů
- BMA algoritmus
- Chien search algoritmus
- Entity pro řídící systém a buffer

Popis vstupů a výstupů BCH dekodéru:

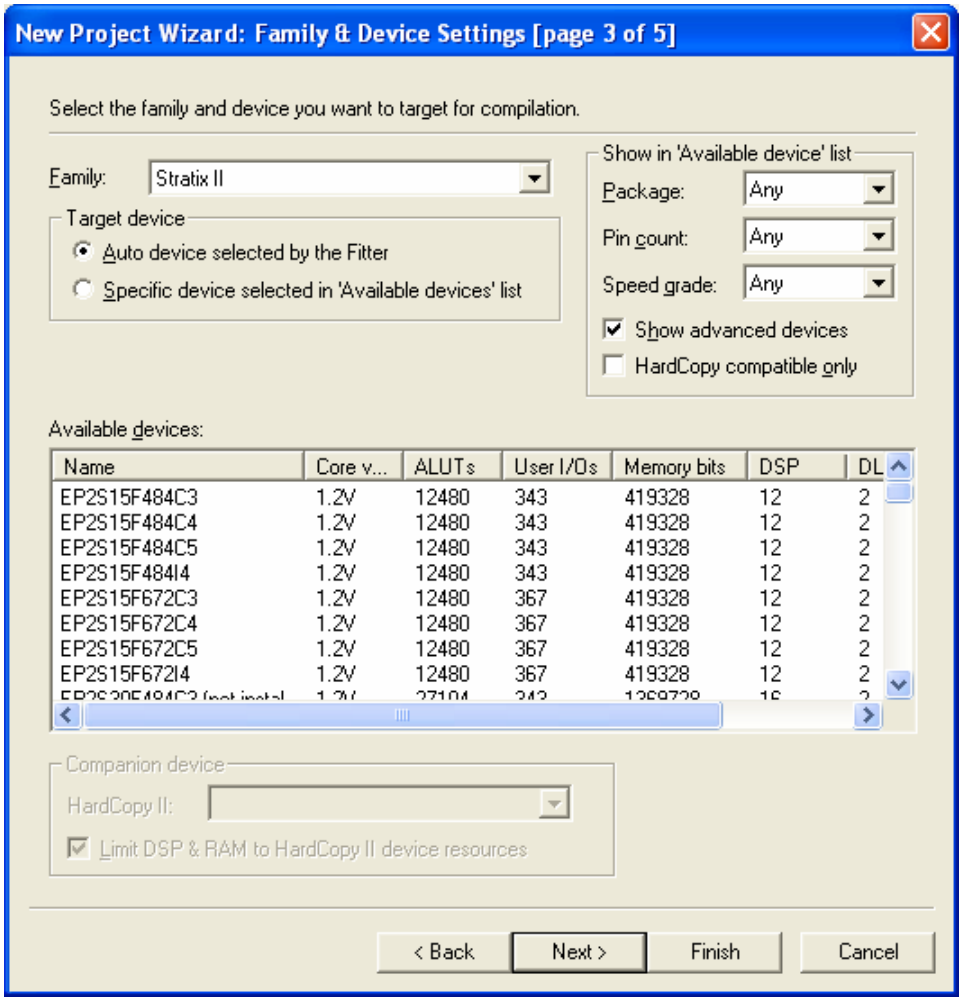
- Reset – vstup - reset dekodéru, možnost synchronizace kodéru a dekodéru
- Din – vstup – vstup dat (kódové slovo vyslané kodérem)
- Dout – výstup – opravená data dekodérem
- Clk – vstup – vstup hodinových impulzů
- Vdout – výstup – indikace stavu, kdy jsou na výstupu připravena již opravená data (Vdout = log1). Může být načteno další kódové slovo ze vstupu.



Obr. 8-3 Dekodér BCH (63, 30)

8.4 Projekt v programu Quartus II

Následná kompilace a syntéza VHDL kódu proběhla ve vývojovém prostředí Altera Quartus II. V tomto prostředí lze vytvořený VHDL kód již přímo zkompilovat pro konkrétní obvod výrobce a odzkoušet časové a funkční parametry tohoto návrhu. Prvním krokem v programu Quartus je vytvoření projektu (každý návrh číslicového obvodu je zde uveden jako projekt). V průběhu vytváření projektu je nutné určit hlavní entitu ve VHDL zápisu (encoder_bch a decoder_bch) a dále vybrat konkrétní obvod, pro který bude návrh optimalizován. Pro kodér i dekodér kódu BCH (63, 30) byla vybrána produktová řada Altera Stratix II a z této řady konkrétní obvod EP2S15F484C3 (jedná se o nejjednodušší obvod z této řady). Obvody řady Stratix II jsou založeny na napájení 1,2 V a 90-nm výrobní technologii. Tyto obvody poskytují až 180 000 logických prvků (LE logic elements), frekvence interních hodin je až 550 MHz, maximální počet použitelných I/O pinů je až 366 (více informací např. v [18]). Ukázka výběru požadovaného obvodu v průběhu vytváření projektu je na oprázku.

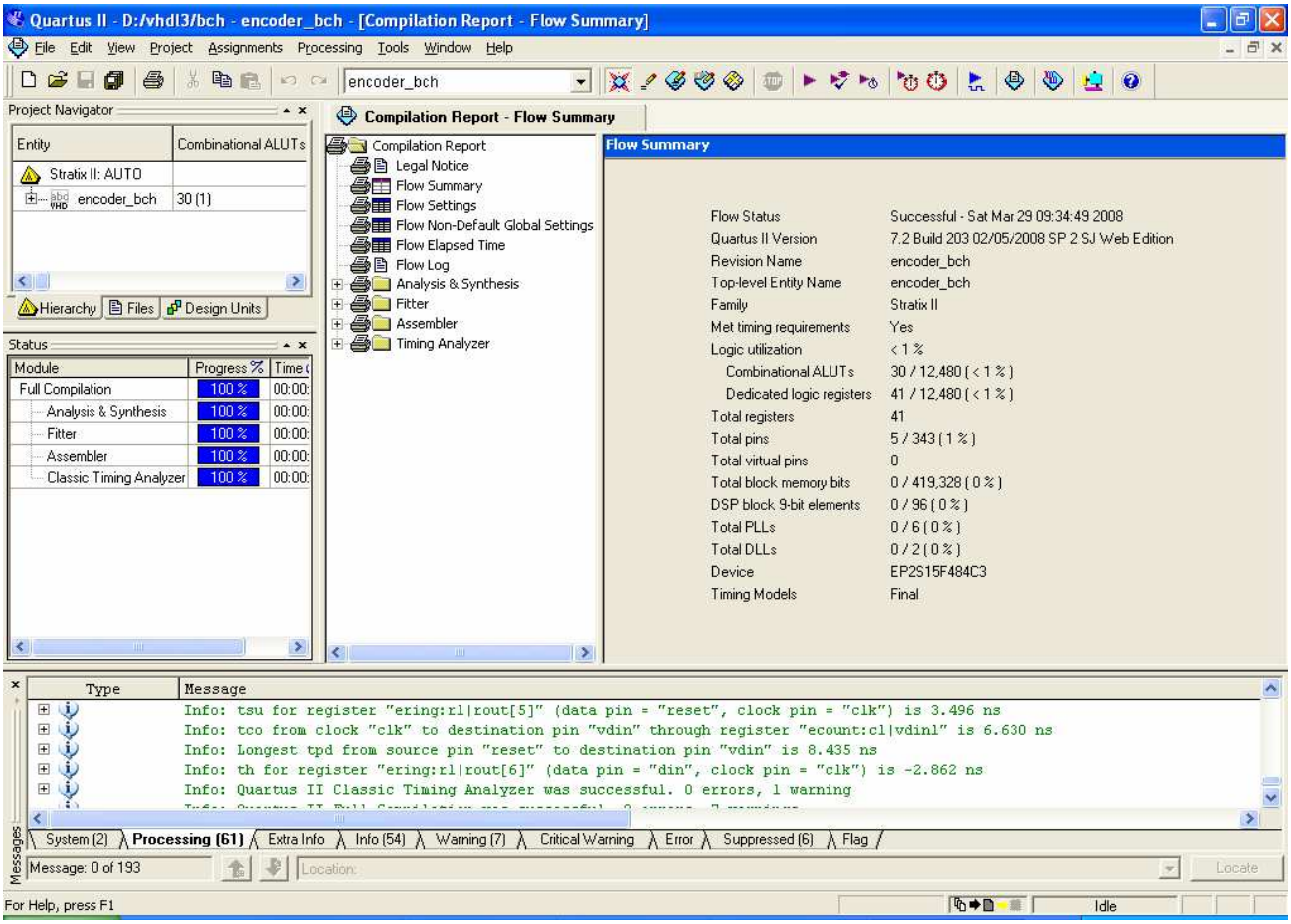


Obr. 8-4 Výběr FPGA obvodu - Quartus

Po vytvoření projektu následuje zpracování VHDL kódu kompilátorem, který provede automaticky analýzu kódu dále se snaží co nejlépe navrhnout vnitřní propojení vybraného FPGA obvodu a v poslední části provede základní časovou analýzu obvodu (viz. Obr. 8-1 návrh FPGA obvodu).

Celý návrh je rozdělen do dvou projektů – jeden pro kodér a druhý pro dekodér BCH kódu. Na přiloženém CD jsou uloženy soubory projektu.

Na následujícím obrázku je vidět výsledná zpráva po kompilaci kodéru.



Obr. 8-5 Kompilace kodéru – Quartus

Požadavky kodéru a dekodéru na hardware:

Tab. 8-1 Požadavky na HW

BCH (63, 30)	IO	Registry	ALUT	Využití IO	f_{max} [MHz]
kodér	EP2S15F484C3	41	30	<1%	500
dekodér	EP2S15F484C4	322	285	3 %	241

IO – integrovaný obvod, LUT – adaptive look-up table

Z tabulky 8-1 je vidět, že dekodér BCH kódu má větší nároky na hardware než kodér.

9. Závěr

Tato práce se zabývá zabezpečením dat pouze pomocí binárních BCH kódů. Tyto kódy jsou schopny zabezpečit binární datový tok proti vzniku nezávislých chyb a patří do skupiny lineárních blokových cyklických kódů. Výhodou této skupiny kódů je volitelnost parametrů (např. délka kódového slova nebo požadovaný počet opravitelných chyb) u těchto kódů je také dobrý poměr mezi počtem informačních znaků a počtem opravitelných chyb. Vlastní zabezpečení přenášených dat se realizuje přidáním zabezpečovacích bitů (redundance) k informačním bitům (užitečná informace). Takto vytvořené kódové slovo se přenáší přes přenosový kanál nebo ukládá na datové uložiště.

V kapitole 1 je rozebrána obecná struktura komunikačního přenosového systému a základní způsoby detekce a opravy chyb. V kapitole 2 jsou uvedeny nejzákladnější informace o BCH kódech a matematické základy týkající se Galoisových těles, které jsou u BCH kódů využívány. Dále je uvedena simulace BCH kódů v Matlabu. Součástí Communications Toolboxu jsou bloky BCH kodéru a dekodéru, kde lze jednoduše měnit parametry BCH kódu. Je porovnán přenos bez a se zabezpečením pomocí BCH kódu přes symetrický binární přenosový kanál s nastavenou chybovostí, dále je uvedena tabulka, kde jsou uvedeny parametry kódů do délky n = 127 bitů.

Požadavkem zadání je vybrat kód, který je schopen opravit t = 6 nezávislých chyb v maximální délce datového toku n = 150 bitů. Toto zadání splňují dva BCH kódy a to kód BCH (63, 30) a BCH (127, 85). První kód BCH (63, 30) je schopen opravit 6 chyb již ve 63 bitech kódového slova. V jednom kódovém slově přenese 30 informačních a 33 redundantních (nutné k zabezpečení) bitů. Druhý kód BCH (127, 85) opraví 6 chyb v kódovém slově délky 127 bitů, přenese 85 informačních a 42 redundantních bitů, má tedy lepší informační poměr, ale také delší kódové slovo při stejném počtu opravitelných chyb jako první kód – je tedy vhodný pro kanály s menší pravděpodobností výskytu chyb. V kapitole 4 je popsáno sestavení vytvářecího mnohočlenu **g**(x) pro kód BCH (63, 30). V kapitole 5 je uveden matematický základ pro kódování BCH kódem. Při kódování se vychází ze zapojení kruhového posuvného registru se zpětnými vazbami a sčítačkami modulo 2 zapojenými podle vytvářecího mnohočlenu. V následující kapitole je uveden matematický základ pro dekódování BCH kódem. Dekódování je poměrně složitější než kódování a vyžaduje složitější matematické postupy. Dekódování lze rozdělit do tří částí: kontrola přijatého kódového slova na

výskyt chyb, nalezení chybového mnohočlenu – lokátoru chyb, oprava chyb. Nejsložitější a výpočetně nejnáročnější je druhý krok – hledání lokátoru chyb. Existuje několik matematických postupů na jeho nalezení. V této práci jsou však stručně uvedeny dva postupy a to Petersonův a Berlekamp – Massey algoritmus. Následná oprava chyb je již poměrně jednoduchá, protože se jedná o binární data, stačí konkrétní bit invertovat.

V kapitole 7 jsou uvedeny možnosti realizace kodeku BCH kódu a základní výběrová kritéria. Nejméně výhodná je realizace pomocí diskrétních integrovaných číslicových obvodů zejména z důvodu složitosti zapojení. Realizace je možná i pomocí mikroprocesoru nebo jako software pro osobní počítač. Většinou by však měl být protichybový kodek integrován do nějakého dalšího systému, který komunikuje s okolím a tím mohou být ovlivněny nároky na rozměry, rychlost, atd. V dnešní době je zřejmě nejzajímavější realizace s využitím programovatelných logických obvodů a to především pomocí programovatelných hradlových polí FPGA. Tyto obvody mají nejobecnější strukturu a poskytují dobrý výkon. Pro další návrh realizace byla vybrána možnost s FPGA obvodem.

V kapitole 8 je nejprve uveden obecný postup při návrhu FPGA obvodu. Návrh probíhá popisem funkce obvodu pomocí VHDL jazyka. Výhodou je, že návrhář nemusí mít speciální znalosti o optimalizaci na úrovni hradel, vnitřních zapojení obvodů a může tuto část ponechat na kompilátoru a syntezátoru (vytvoření netlistu), který lze získat přímo od výrobce programovatelných logických obvodů. Pro návrh jsem použil vývojové prostředí od firmy Altera – Quartus WebPack, které je možné získat po zaregistrování zdarma. Kodér i dekodér může být realizován stejným obvodem s označením EP2S15F484C3 z řady Altera Stratix II. Z výsledné zprávy po kompilaci je vidět, že dekodér je opravdu hardwarově náročnější než kodér. Pro dekodér je využito 322 registrů a maximální rychlost je odhadnuta na f_{max} = 241 MHz - funkce dekodéru využívá potenciál obvodu přibližně na 3 %, pro kodér je využito 41 registrů a rychlost může být až 500 MHz - funkce kodéru využívá potenciál obvodu přibližně z 1 %. Z hodnot využití obvodu je zřejmé, že FPGA obvody v dnešní době poskytují velmi zajímavé možnosti aplikací. Domnívám se, že zadání práce bylo splněno podle požadavků. Další testování funkce kodeku by se již mělo realizovat přímo v obvodu FPGA na vývojové desce.

Použitá literatura

[1] WALLACE, H. *Error detection and correction using the BCH code* [online]. c2001. [cit. 20. 2. 2008]. Dostupné na: http://www.aqdi.com/bch.pdf,

[2] VLČEK, K. *Komprese a kódová zabezpečení v multimediálních aplikacích.* BEN Technická literatura, Praha 2004, ISBN: 80-7300-134-9,

[3] NĚMEC, K. *Protichybové kódové zabezpečení s Bose-Chauhury-Hocquenhemovými kódy* [online].[cit. 20. 2. 2008]. Publikace v internetovém magazínu Elektrorevue. Dostupné na: http://www.elektrorevue.cz/clanky/05015/,

[4] NĚMEC, K. *Datová komunikace.* skriptum VUT-Brno, VUTIUM 2000,

[5] ADÁMEK, J. *Kódování*, skriptum ČVUT, SNTL Praha 1986,

[6] COSTELLO, J., LIN, S. *Error Control Coding: Fundamentals and Application*, Prentice-Hall 1983, ISBN: 978-0130426727

[7] ČÍKA, P. *Protichybové zabezpečení BCH kódem* [online]. [cit. 28. 2. 2008]. Publikace v internetovém magazínu Elektrorevue.
Dostupné na: http://www.elektrorevue.cz/clanky/06015/,

[8] NĚMEC, K. *Majoritní dekódování blokových kódů* [online]. Publikace v internetovém magazínu Elektrorevue. Dostupné na: http://www.elektrorevue.cz/clanky/04041/,

[9] KOLOUCH, J. *Elektronické skriptum* [online]. [cit. 28. 2. 2008]
Dostupné na: http://www.urel.feec.vutbr.cz/~kolouch/pld/ramy02.html,

[10] ATMEL, *Microcontrollers Atmel – product ATmega32* [online].
Dostupné na: http://www.atmel.com,

[11] XILINX, *Programmable logic devices* [online].
Dostupné na: http://www.xilinx.com,

[12] ALTERA, *Using Programmable Logic Device* [online].
Dostupné na: http://www.atmel.com/dyn/resources/prod_documents/DOC0485.PDF,

[13] WILEY, J., *Essentials of Error-Control Coding* , John Wiley & Sons Publisher 2006, ISBN: 047002920X,

[14] TODD, K., *Error correction coding: mathematical methods and algorithms*, John Wiley & Sons Publisher 2005, ISBN: 0-471-64800-0,

[15] MICHELI, G., Ernst, R., *Readings in Hardware / Software Co-Design*, Morgan Kaufmann Published 2001, ISBN: 1558607021,

[16] PINKER, J., POUPA, M., *Číslicové systémy a jazyk VHDL*, Nakladatelství BEN Praha 2006, ISBN: 80-7300-198-5,

[17] ALTERA, *Introduction to the Quartus II Software* [online].
Dostupné na: http://www.altera.com/literature/manual/intro_to_quartus2.pdf,

[18] ALTERA, *Quartus II Handbook* [online].
Dostupné na: http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf,

[19] ALTERA, *Stratix II Introduction* [online].
Dostupné na: http://www.altera.com/literature/hb/stx2/stx2_sii51001.pdf,

[20] JAMRO, E., The design of a VHDL based synthesis tool for BCH codes, [online]
Dostupné na: http://home.agh.edu.pl/~jamro/bch_thesis/bch_thesis.html.

Seznam obrázků

Obr. 1-1 Struktura komunikačního kanálu 12
Obr. 3-1 Schéma BCH dekodéru v Matlabu 20
Obr. 3-2 Schéma BCH kodéru v Matlabu 20
Obr. 3-3 Zapojení přenosového systému bez BCH kodeku 21
Obr. 3-4 Zapojení přenosového systému s BCH kodekem 21
Obr. 5-1 Rozložení bitů v kódovém slově 27
Obr. 5-2 Posuvný kruhový registr 27
Obr. 5-3 Kodér kódu BCH (63,30) pomocí kruhového posuvného registru 28
Obr. 6-1 Schéma dekódování 30
Obr. 8-1 Proces návrhu FPGA obvodu 42
Obr. 8-2 Kodér BCH (63, 30) 44
Obr. 8-3 Dekodér BCH (63, 30) 45
Obr. 8-4 Výběr FPGA obvodu - Quartus 46
Obr. 8-5 Kompilace kodéru – Quartus 47

Seznam tabulek

Tab. 2-1 AND a XOR v tělese Z_2 16
Tab. 3-1Výsledky Matlab 22
Tab. 4-1 BCH kódy pro t = 6 chyb a n = 150 bitů 23
Tab. 4-2 Nerozložitelné mnohočleny nad GF(2^6) 25
Tab. 7-1 Kritéria pro výběr realizace 40
Tab. 8-1 Požadavky na HW 47

Seznam zkratek

ARQ	**A**utomatic **R**epeat re**Q**uest
BCH	**B**ose-**C**haudhuri-**H**ocquenghem
BER	**B**it **E**rror **R**ate
CPLD	**C**omplex **P**rogrammable **L**ogic **D**evice
FEC	**F**orward **E**rror **C**orrection
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FTP	**F**ile **T**ransfer **P**rotocol
HDL	**H**ardware **D**escription **L**anguage
LFSR	**L**inear **F**eedback **S**hift **R**egister
PLD	**P**rogrammable **L**ogic **D**evice
VHDL	**V**ery (High Speed Integrated Circuit) **H**ardware **D**escription **L**anguage