VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE SAMOOPRAVNÝCH KÓDŮ PRO 100 GB/S ETHERNET

IMPLEMENTATION OF SELF-CORRECTING CODES FOR 100 GB/S ETHERNET

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN VELECKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. LUKÁŠ KEKELY**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19862/2016/xvelec07

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Velecký Jan**
Obor: Informační technologie
Téma: **Implementace samoopravných kódů pro 100 Gb/s Ethernet**
Implementation of Self-Correcting Codes for 100 Gb/s Ethernet
Kategorie: Návrh číslicových systémů

Pokyny:

1. Seznamte se s existujícími samoopravnými kódy používanými pro datové přenosy.
2. Seznamte se s technologií FPGA a s protokolem 100 Gb/s Ethernet, predevším s jeho variantou 100GBASE-SR4 využívající opravné kódování Reed-Solomon.
3. Navrhněte vrstvu RS-FEC pro standard 100GBASE-SR4.
4. Navržené moduly vysílací části implementujte v FPGA jako rozšíření stávajícího řešení 100GBASE-LR4 a ověřte jejich funkční parametry.
5. Diskutujte možnosti využití modulu na FPGA čipech UltraScale a pro rychlejší standard Ethernet 400 Gb/s.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kekely Lukáš, Ing.**, UPSY FIT VUT
Konzultant: Friedl Štěpán, Ing., CESNET
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Práce se zabývá návrhem ucelené RS-FEC vrstvy pro 100Gb/s Ethernet dle standardu IEEE 802.3-2015 včetně kódovacího a dekódovacího obvodu Reed-Solomonova kódu. Text objasňuje matematický aparát konečných těles, lineárních blokových kódů, cyklických kódů a zejména samotných Reed-Solomonových kódů pro použití v návrhu.

Návrh vysílací části RS-FEC vrstvy byl přizpůsoben pro implementaci v síťových kartách COMBO využívajících FPGA čipy Xilinx Virtex-7 a realizován ve VHDL. Kódovací obvod byl v několika krocích zoptimalizován – co se týče požadavků na zdroje FPGA a délky trvání syntézy VHDL kódu. Snížení nároků na zdroje se docílilo zejména využitím vlastností cyklických kódů umožňující zřetězení. Doba syntézy pak vytvořením logiky kódovacího obvodu na úrovni hradel ve vlastní režii. Výsledná implementace byla testována v simulaci a je dostatečně zoptimalizována, aby mohla být použita při implementaci Ethernetu na FPGA čipu.

Jak návrh, tak implementaci je možné modifikovat pro 400Gb/s Ethernet – v době návrhu ještě oficiálně neexistujícího.

Abstract

The thesis deals with the design of entire RS-FEC layer for the 100 Gb/s Ethernet according to IEEE 802.3-2015 standard including Reed-Solomon encoder and decoder. Text clarifies mathematical basis of finite fields, linear block codes, cyclic codes and particularly Reed-Solomon codes used in design.

Design of RS-FEC layer transmit side has been adjusted for implementation in COMBO network cards which use Xilinx Virtex-7 FPGA and realized in VHDL. Encoder has been optimized in several steps – as for FPGA resource usage and as for VHDL code synthesis duration. Reduction of resource usage has been achieved by using pipelining thanks to properties of cyclic codes. Synthesis duration then by creating logic of encoder on gate level on its own. Resulting implementation has been tested in simulation and it is optimized enough for usage in FPGA for Ethernet implementation.

It is possible to adapt both design and implementation for 400Gb/s Ethernet which does not exist yet at the time of design.

Klíčová slova

Reed-Solomonovy kódy, RS-FEC, Galoisova tělesa, konečná tělesa, FPGA, 100GBASE, LR4, SR4, CR4, KP4, oprava chyb, protichybové kódování

Keywords

Reed-Solomon codes, RS-FEC, Galoise fields, finite fields, FPGA, 100GBASE, LR4, SR4, CR4, KP4, error correction, error-control coding

Citace

VELECKÝ, Jan. *Implementace samoopravných kódů pro 100 Gb/s Ethernet*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kekely Lukáš.

Implementace samoopravných kódů pro 100 Gb/s Ethernet

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Lukáše Kekelyho a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal. Další odborné informace mi poskytl Ing. Štěpán Friedl ze sdružení CESNET.

. . . . . .
Jan Velecký
17. května 2017

Poděkování

Rád bych tímto poděkoval svému vedoucímu, Ing. Lukáši Kekelymu, za odborné vedení a poskytnuté rady při psaní této práce. Rovněž bych rád poděkoval Ing. Štěpánu Friedlovi za jeho pomoc při studiu problematiky, implementačních pracích i za poskytnuté potřebné znalosti prostředí, kde má být výsledek práce upotřeben.

Obsah

Kapitola 1

Úvod

V dnešní době je běžné, že chceme přenášet velké objemy dat a navíc se nám nechce ani čekat. To klade vysoké nároky na techniku, která to zajišťuje. Dokonce takové, že i přenosová média dříve jednoduše považovaná za spolehlivá přestávají se současnými nároky spolehlivá být. Projevuje se to tak, že při přenosech dochází k chybám – doručená data neodpovídají těm odeslaným. To se řeší tzv. protichybovými kódy umožňujícími získat správná data i v případě takovéhoto chybného přenosu. A tyto kódy se tak nyní prosazují i tam, kde dříve potřebné nebyly. Tedy i v optických a krátkých metalických spojích standardu Ethernet, jež se používají pro budování internetových sítí.

Dnešní varianty Ethernetu umožňují přenosové rychlosti až 100 Gb/s. Takové rychlosti nejsou dnes určeny pro domácí uživatele, ale používají je např. poskytovatelé infrastruktury Internetu k propojení mezi sebou. Nový standard zavádí poprvé protichybové Reed-Solomonovy kódy. V době psaní této práce se připravují varianty s rychlostí 400 Gb/s, kde se budou používat rovněž.

Cílem této práce je navrhnout řešení té části varianty 100GBASE-SR4 100Gb/s Ethernetu, která se věnuje protichybovému kódování – RS-FEC vrstva. Dále v rámci práce vysílací stranu implementovat v jazyku VHDL (sloužícímu pro popis hardwaru) pro programovatelná hradlová pole (FPGA). Při implementaci se bude vycházet z podobnosti k variantě LR4, která se liší především chybějící protichybovou vrstvou. Toho se využije tak, že implementace RS-FEC vrstvy bude vložena do již existující LR4 implementace, čímž vznikne varianta SR4. Existující implementace bude získána v rámci výzkumného projektu Liberouter[1] a výsledná v něm využita.

Ačkoliv existují implementace RS-FEC kódování pro platformu FPGA, většinou nedosahují požadované propustnosti nového standardu, mají odlišné vlastnosti, nejsou optimalizovány, často nejsou volně k použití, případně se jedná o velmi nákladné licence nebo mají nedostatečnou dokumentaci. Toto řešení ale navíc bude nabízet implementaci ucelené jednotky RS-FEC vrstvy daného standardu, nikoliv pouze samotné protichybové kódování, které je pouze částí funkce této vrstvy. Modul bude s úpravami, případně pouhou změnou konfigurace RS kódování, použitelný i jako základ pro vytvoření řešení jiných podobných variant Ethernetu.

V 2. kapitole této práce je vysvětleno, co to jsou datové přenosy, kódování a především protichybové kódy, aby se zavedly důležité pojmy z této oblasti. Následně se zabývá matematickou teorií protichybových kódů (matematickým aparátem konečných těles v podkapitole 2.2.1, Reed-Solomonovy kódy v podkapitole 2.2.4) a jejich reprezentací. Toto ilustruje na příkladech jednoduchých kódů. Kapitola 3. se zabývá technologickou stránkou věci – stručně popisuje cílovou platformu FPGA, popisuje Ethernet a podrobněji jeho konkrétní variantu, jíž se zabývá tato práce. Kapitola 4. řeší návrh a implementaci na straně odesílatele a 5. návrh strany příjemce. V poslední kapitole je shrnutí výsledků a popis dalšího vývoje v kontextu budoucího rychlejšího Ethernetu.

[1] https://www.liberouter.org/

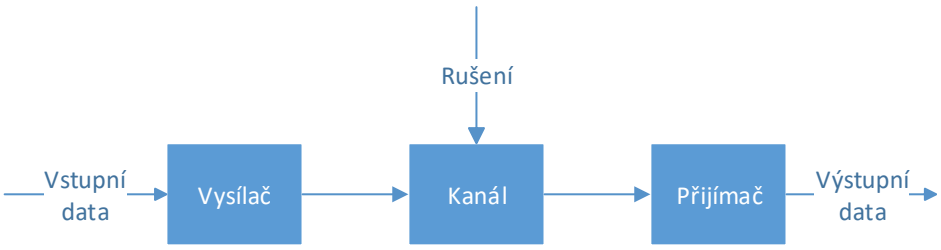
Kapitola 2

Datové přenosy a jejich spolehlivost

V této kapitole budou vymezeny pojmy související s datovými přenosy a jejich vlastnostmi, dále používané techniky kódování – zejména ty zvyšující spolehlivost přenosu.

Data jsou posloupnost symbolů obsahující informační hodnotu, kterou je možno uchovávat, přenášet či interpretovat. Data, která ve formě signálu fyzicky přenášíme *datovým kanálem* mezi zařízeními (různými místy/koncovými body) se nazývají *zpráva* a tato činnost *datovým přenosem*. Koncovými body přenosu jsou *vysílací* (odesílací, zdrojová) a *přijímací* (cílová) *strana* (značí se obvykle Tx a Rx z angl. Trasmit a Receive), zkráceně odesílatel a příjemce, nebo také zdroj a spotřebič informace. Přenášejí-li si zařízení mezi sebou data, probíhá mezi nimi *datová komunikace* – zařízení si sdělují informaci.

Nákres datového přenosu je na obr. 2.1. Přenášená data jsou na datovém kanálu vyjádřena hodnotou fyzikální veličiny – *signálem* (např. napětí). Tato hodnota je na datový kanál vystavena na zdrojové straně *vysílačem* (budičem) (angl. transmitter) a na cílové straně je z kanálu čtena *přijímačem* (snímačem). Působením vnějších vlivů prostředí (rušením neboli interferencí) může docházet na některých přenosových médiích k pozměnění přenášeného signálu do takové míry, že *výstupní (přijatá) data* získaná čtením signálu přijímačem budou mít odlišný význam od *vstupních dat* vyslaných vysílačem – dojde tedy k příjmu *chybných dat*.



Obrázek 2.1: Blokové schéma datového přenosu

Datovou komunikaci můžeme charakterizovat podle několika kritérií:

- Sériovost/paralelnost rozhraní – zda jsou data odesílána přes jedno/více přenosových médií
- Synchronnost/asynchronnost přenosu – synchronizace je/není průběžně přenášena signálem
- Kontinuita přenosu – data jsou odesílána nárazově, jakmile jsou k dispozici, nebo neustále s tím, že některé odesílané sekvence nemají informační význam
- Směr komunikace – přenosy probíhají jednosměrně (simplexně), nebo obousměrně (duplexně). V případě obousměrného přenosu může být sloučena funkce přijímače a vysílače do jednoho

prvku budič-přijímač (angl. transceiver z TRANSmitter a reCEIVER, dále bude užíváno angl. pojmenování)

- Počet příjemců – dvoubodová (jeden příjemce; angl. point-to-point) / vícebodová (více příjemců; angl. point-to-multipoint) komunikace
- Spolehlivost média – zda je přenos dat náchylný na vznik chyb
- Spojitost kanálu [1]
  - spojitý – přenáší se analogová data
  - diskrétní – v čase nebo hodnotách nespojitý
  - digitální (diskrétní v čase i hodnotách) – hodnoty signálu se mění skokově ve stanovené okamžiky. Obvykle je digitální kanál binární, v takovém případě má signál pouze 2 možné významy. Existují i ternární kanály, aj.

Vzhledem k zaměření práce se dále budeme zabývat pouze asynchronními kontinuálními dvoubodovými binárními přenosy.

2.1 Kódování datových přenosů

Vstupní data přicházející ze zdroje dat tvoří *datový proud* (angl. datastream), který není možné vždy přímo převádět na signál vysílačem z různých důvodů uvedených dále. Proto se datový proud před odesláním přizpůsobuje možnostem datového kanálu – provádí se překódování a vzniká *kódový proud* (angl. codestream) a až ten je vysílán.

Definice 2.1.1. *Kód* je postup převedení zprávy do její jiné ekvivalentní reprezentace. Jedná se o převod prvků zprávy (symbolů) mezi vstupní a výstupní abecedou. *Abeceda* je množina symbolů, kterými prvky zprávy mohou být. *Kódování* je provádění převodu.

Kódování je vratný proces, můžeme jej tedy odlišovat podle směru na *(za)kódování* (angl. encoding) – překódování zdrojové zprávy na cílovou a *dekódování* (angl. decoding) – překódování zpět na zdrojovou zprávu. Pokud procesem kódování může dojít ke ztrátě informace (obecně ke změně významu zprávy), nazývá se tento proces *transkódování* (angl. transcoding). Opačný proces se nazývá *transdekódování* a je nutné se v něm vypořádat s možnou chybějící informací. Např. převod celého čísla v desítkovém zápisu do BCD kódu je kódování. Kdežto odstranění diakritiky z textu je transkódováním.
Důvody vedoucí k užití překódování u datových přenosů:

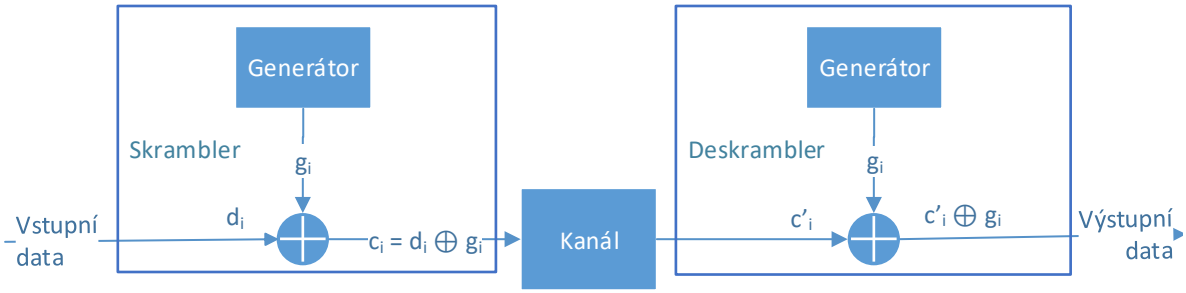
- Nedostatečný počet úrovní signálu datového kanálu vzhledem k počtu symbolů vstupních dat. Překódování umožňuje přenesení tím, že n prvků zprávy je přeneseno jako m prvků kódu, kde $m > n$.
- Nízká efektivita přenosu, pokud často přenášené sekvence mají stejný počet symbolů jako sekvence vyskytující se méně často. Nadbytečnost se omezuje použitím nerovnoměrného překódování, kdy se sekvence překódují na délku nepřímo úměrnou četnosti výskytu. Používají se prefixové kódy (např. Huffmanův kód).
- Potřeba snížit frekvenci překlápění signálu. Např. metoda MFM.
- Zabránění periodicit v datovém toku, které můžou způsobit přenášení energie v určitém frekvenčním pásmu. Jednou z technik je použití tzv. skramblerů.

- Omezení délky konstantní úrovně signálu (RLL – Run Length Limited). Např. u přenosu vodičem k dosažení nulové stejnosměrné složky napětí. Např. kódování 8b/10b.
- Vsazení dostatečného množství změn signálu pro umožnění získání synchronizace. Např. metoda vzorkování dvojnásobným kmitočtem (FM).
- Zvýšení spolehlivosti přenosu po nespolehlivém kanálu – zabezpečení proti chybám. Používají se detekční kódy (např. paritní kód) a opravné kódy (např. Hammingův kód).
- Znemožnění čtení dat přenášených po datovém kanálu jiným zařízením než adresovaným. Tento způsob kódování se nazývá šifrování.

Pro tuto práci jsou podstatné kódy zabezpečení proti chybám a skramblery, proto se nimi budou zabývat následující kapitoly.

2.1.1 Skramblování

Skramblování (podle [1], kap. 4.1) datového proudu mění jeho charakter tím, že pseudonáhodně překlápí jeho hodnoty. Získává tak užitečné vlastnosti, kterými jsou: rozrušení periodicit v něm, dostatečné množství změn úrovní vyslaného signálu (což je užitečné k udržení synchronizace u příjemce), dosažení parametru RLL. V datových přenosech provádí skramblování odesílatel a příjemce *deskramblování*, což je proces získání původních dat. Scramblování v tomto kontextu je ilustrováno na obrázku 2.2. Odesílatel vysílá na datový kanál zaskramblovaná (vstupní) data. Ta (pokud nedošlo při přenosu k chybě) po deskramblování příjemcem (výstupní data) odpovídají datům vstupním. Komponenta, která provádí (de)skramblování se nazývá (de)skrambler. Skrambler generuje posloupnost pseudonáhodných čísel g_i rozhodujících o tom, zda bude aktuální vstupní symbol skrambleru v jeho výstupu překlopen $c_i = d_i \oplus g_i$[1]. Deskrambler generuje synchronně tutéž posloupnost g_i a používá je k překlopení přijatých symbolů. Příjemce tak získává symbol $d'_i = c'_i \oplus g_i$. A pokud přijatý symbol c'_i odpovídá vyslanému c_i, pak $c'_i \oplus g_i = d_i \oplus g_i \oplus g_i = d_i$.



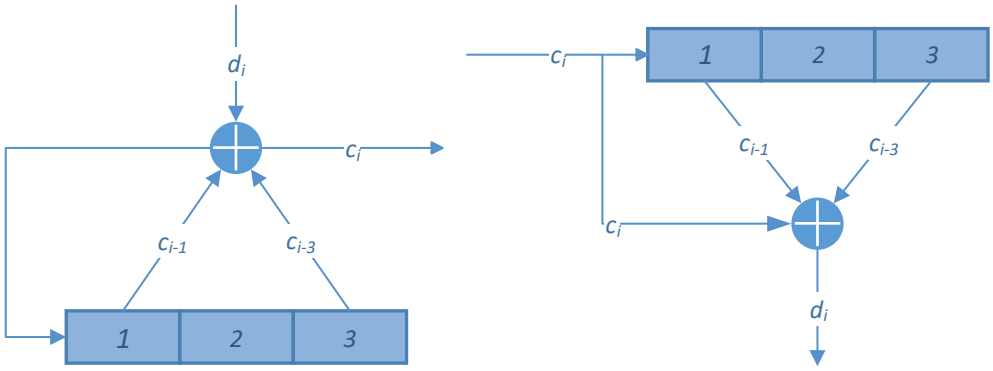
Obrázek 2.2: Blokové schéma skrambleru–deskrambleru a jeho použití

Jako generátor posloupnosti skrambleru se často používá lineárně-zpětnovazební posuvný registr (LFSR). Skramblování se dělí podle způsobu naplnění tohoto registru na:

- Synchronní (angl. reset scrambler) – Je nutné zajistit synchronizaci včetně stejné výchozí hodnoty registrů obou skramblerů. Deskrambler provádí v tomto případě identickou činnost jako skrambler.
- S vlastní synchronizací (angl. self-synchronizing scrambler) – Skrambler plní registr svým výstupem (zpětná vazba). Deskrambler ho plní svým vstupem (dopředná vazba) – naplněním dojde k synchronizaci.

[1] Symbol $\oplus$ v textu i v obrázcích značí operaci součet mod 2.

Zapojení zpětné vazby se často zapisuje ve formě *vazebního mnohočlenu V(x)*, např. $V(x) = 1 + x^{-1} + x^{-3}$, který určuje které stupně LFSR se účastní zpětné vazby. Schéma tohoto skrambleru je na obrázku 2.3. ([2], kap. 11.2.3)



Obrázek 2.3: Sebesynchronizační skrambler (vlevo) a deskrambler

2.2 Ochrana dat proti chybám

Pro zabezpečení datových přenosů proti chybám se používají tzv. *bezpečnostní kódy* (také protichybové). Tato podkapitola rozebírá principy a vlastnosti těchto kódů, čerpá z ([1], kap. 4.2). Následovně budou vybrané bezpečnostní kódy naznačeny.

Nezabezpečená zpráva je tvořená pouze *informačními symboly*. Bezpečnostní kódování zavádí do odesílané zprávy takové pravidla, díky kterým je možno na straně příjemce rozeznat (příp. i opravit) poškozenou zprávu tím, že přijatá zpráva daným pravidlům nevyhovuje. To se neobejde bez rozšíření odesílané zprávy o *zabezpečovací symboly* (čili kontrolní a často také nepřesně označované jako paritní[2]), avšak informačně nadbytečné (redundantní). Tímto se zvyšuje spolehlivost přenosu v korelaci s mírou redundance. Bezpečnostní kódy se dělí na 2 základní kategorie podle toho, zda sebou zakódovaná zpráva nese informace dostatečné také pro to, aby se příjemce v případě detekce chyby samostatně mohl pokusit o opravu přijaté zprávy:

- Detekční kódy (EDC z angl. Error Detection Codes):

  Umožňují na straně příjemce pouze detekci chyby, nikoliv její nápravu (případně k tomu nejsou zamýšleny). Ta se děje pomocí techniky zvané *ARQ* (z angl. Automatic Retransmission reQuest), kdy příjemce v případě přijetí chybné zprávy odesílateli pošle žádost o nové zaslání dat. Nevýhodou tohoto řešení je, že zanáší do přenosu zpoždění, tím pádem nelze využít pro systémy reálného času. Tyto kódy ale nemusí být používány přímo pro zvyšování spolehlivosti, ale např. pouze pro monitorování kvality přenosu, nebo k zahazování poškozených zpráv, aby nedošlo k jejich (špatné) interpretaci.

  Nejjednodušší detekční kód je *parita*, kdy se do binární zprávy přidává kontrolní symbol (paritní bit), tak aby zpráva vč. parity obsahovala sudý, nebo lichý počet jedničkových bitů – tzv. sudá, nebo lichá parita. ([2], kap. 4.1) Např. zpráva 0010110 bude zabezpečena sudou paritou k vyslání jako 0010110 1. Dalším zástupcem této kategorie je kupř. CRC.

- Opravné kódy (ECC z angl. Error-Correcting Codes, jinak také korekční, samoopravné):

[2] Paritní symbol je název kontrolního symbolu paritního kódu, který je popsán dále v textu.

Umožňují jak detekci, tak i eventuální opravu poškozené zprávy – může být detekována neopravitelná chyba, proto se i tyto kódy někdy kombinují s ARQ. Nicméně primární technika opravy chyby pro tyto kódy je *FEC* (dopředná oprava chyb, z angl. Forward Error Correction), jež spočívá ve vytvoření takových kontrolních symbolů odesílatelem, které slouží pro následnou opravu chyb u příjemce bez nutnosti zpětné vazby na odesílatele. Proto je zde míra redundance obecně vyšší než u detekčních kódů a také kódování je složitější. Zástupci těchto kódů jsou kupř. Hammingův, Golayův, BCH, Reed-Solomonův,...

Dále se dají bezpečnostní kódy dělit podle toho, zda se kontrolní symboly vkládají průběžně:

- Konvoluční kódy (spojité, řetězové) – kontrolní symboly jsou vypočítávány a vkládány do posloupnosti informačních symbolů průběžně
- Blokové kódy – zpráva je nejprve rozdělena na nezávislé bloky předem dané velikosti, pro které jsou kontrolní symboly vypočteny a k nim připojeny

A podle způsobu umístnění kontrolních symbolů:

- Systematické kódy – kód přesně určuje, na kterých pozicích posloupnosti prvků zprávy se nachází kontrolní symboly a na kterých informační. Parita je systematický kód
- Nesystematické kódy – nelze v nich rozlišit kontrolní a informační symboly

Nejvýznačnější zabezpečovací kódy jsou systematické blokové kódy a často se označují jako *kód*(n, k), kde n určuje velikost zabezpečeného bloku zprávy v symbolech a k určuje, kolik symbolů z nich je informačních (tj. velikost nezabezpečené části zprávy). Počet kontrolních symbolů je $r = n - k$. ([2], kap. 3) Kód má celkem c^n variant slov (bloků), které může příjemce v rámci zprávy obdržet, kde c představuje kardinalitu abecedy přenosu, čili c je 2 pro binární přenosy. Z nich však pouze c^k variant jsou tzv. *kódové slova*, což jsou takové zabezpečené slova, kde platí, že kontrolní symboly správně odpovídají informačním symbolů dle definice kódu. Všechny ostatní varianty jsou *nekódová slova*, jejichž přijetí znamená, že při přenosu došlo k chybě.

Z těchto parametrů můžeme stanovit hodnotící kritérium *redundance kódu*

$$R = (n - k)/n \ \ [\%]$$

udávající relativní počet symbolů, které je nutno do zprávy přidat k zajištění zabezpečení a dá se tak hodnotit úspornost daného kódu. Dalším kritériem může být poměr počtu kódových a nekódových slov (kódová:nekódová), jenž se nazývá *CNC* (Code to NonCode ratio). Např. CNC paritního kódu je 1:1. [3]

Počet symbolů, v kterých se liší 2 kódová slova, se nazývá *Hammingova vzdálenost*, značí d a obecně může být různá pro různé dvě dvojice kódových slov zkoumaného kódu. Zásadní význam však má *minimální Hammingova vzdálenost* značená d_{min}, jež je minimální vzdálenost mezi libovolnými 2 kódovými slovy a charakterizuje kód jako takový, proto se nazývá také *kódová vzdálenost*. Určuje ke kolika změnám různých symbolů alespoň musí dojít v kódovém slově, aby se potencionálně stalo jiným kódovým slovem a tím pádem bylo příjemcem interpretované jako bezchybně přijaté, ačkoliv se liší od vyslaného. V tabulce 2.1 je zobrazeno, kolik chyb kód dokáže detekovat, nebo opravit dle jeho kódové vzdálenosti.

Protože při datových přenosech se vyskytují různé druhy chyb (a jejich kombinace):

- jednoduchá chyba – jediná chyba v zabezpečeném bloku dat,
- nezávislé chyby – vícenásobné chyby ve velké vzdálenosti od sebe ve zprávě,

d_{min}	detekovatelné chyby	opravitelné chyby	kategorie
1	0	0	typicky nijak nezabezpečená zpráva
2	1	0	SED (Single Error Detection)
3	2	1	SEC (Single Error Correction)
4	3	1	SEC-DED
5	4	2	DEC

Tabulka 2.1: Klasifikace kódů dle kódové vzdálenosti [3]

- shlukové (závislé) chyby – vícenásobné chyby koncentrované ve zprávě poblíž sebe,

je tedy možno i bezpečnostní kódy dělit podle toho, zda jsou zaměřeny na zabezpečení přenosu proti nezávislým chybám, nebo proti shlukům chyb. Přičemž galvanické vedení obvykle vytváří shlukové chyby a naopak jiné druhy chyb v něm nejsou časté. ([2], kap. 2.3.1)

2.2.1 Galoisova tělesa

Galoisova tělesa (též konečná tělesa, angl. Galoise/finite fields, zkracováno jako GF) jsou matematickým základem pro mnoho kódů. Z toho důvodu je jim věnována tato podkapitola. Informace pro tuto podkapitolu jsou čerpány z ([2], kap. 4.5.1), ([4], kap. 2.2), ([5], kap. 4.).

Definice 2.2.1. ([6], kap. 1.1) *Těleso* je algebra $(F, +, \cdot)$. Nad množinou F jsou definovány operace součtu $+$ a součinu $\cdot$, a musí pro ně platit axiomy tělesa (pro všechny $a, b, c \in F$):

1. Komutativita operací:
   $a + b = b + a \qquad a.b = b.a$
2. Asociativita operací:
   $(a + b) + c = a + (b + c) \qquad (a.b).c = a.(b.c)$
3. Existence nulových (neutrálních) prvků 0 pro sčítání a 1 pro součin:
   $a + 0 = a \qquad a.1 = a$
4. Existence inverzních prvků $-a$ pro sčítání a b^{-1} pro součin ($b \neq 0$):
   $a + (-a) = 0 \qquad b.b^{-1} = 1$
5. Distributivní zákon:
   $a.(b + c) = a.b + a.c$
6. Netrivialita:
   $0 \neq 1$

Pokud je kardinalita F konečná, jedná se o těleso konečné.

Množina F s operací nad ní definovanou splňující axiomy 1 až 4 se nazývá *komutativní grupa* (také Abelova). Existují tudíž alternativní definice, které axiomy 1–4 vyjadřují tak, že $(F, +)$ i $(F - 0, \cdot)$ tvoří komutativní grupu.

Operace součinu $a \cdot b$ se běžně zapisuje jako ab. Dále často definujeme v tělese rozdíl $-$ a podíl $/$ pomocí inverzních prvků:

- rozdíl pomocí aditivní inverze $a - b = a + (-b)$,
- podíl pomocí multiplikativní inverze $a/b = a.b^{-1}$.

Protože 0 nemá multiplikativní inverzi, není definováno ani dělení pro $b = 0$.

Definice 2.2.2. *Konečné těleso* $\mathbb{Z}_p$ (také těleso zbytkových tříd, nebo prvočíslené těleso, jinak značeno také $GF(p)$, nebo $\mathbb{F}_p$) je těleso $(\{0, 1, ..., p-1\}, +, \cdot)$, kde p je prvočíslo. Prvky tohoto tělesa se nazývají *množina zbytkových tříd* – je tvořena zbytky přirozených čísel po dělení číslem p. Operace jsou definovány jako sčítání a součin modulo p.

Operace binárního tělesa $\mathbb{Z}_2$ součtu i rozdílu ($a \pm b \bmod 2$) je nonekvivalence $\oplus$ a násobení ($a.b \bmod 2$) je konjunkce. Toto se také někdy nazývá *aritmetika mod 2*, obecně aritmetika mod p. V tabulce 2.2 je ukázka počítání v algebře ternárního tělesa $\mathbb{Z}_3$.

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

·	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Tabulka 2.2: Operace v tělese $GF(3)$

Definice 2.2.3. *Galoisovo těleso* $GF(p^r)$ je označení tělesa o konečném počtu $n = p^r$ prvků (p je prvočíslo nazývané *charakteristika tělesa*). r je stupněm rozšíření tělesa $\mathbb{Z}_p$, tím se myslí, že prvky tělesa jsou nyní r-tice koeficientů ze $\mathbb{Z}_p$:

$$GF(p^r) = \{(a_0, a_1, ..., a_{r-1}); a_i \in \mathbb{Z}_p; i = 0..r-1\}$$[3]

Častěji se prvky tělesa $(a_0, a_1, ..., a_{r-1})$ vyjadřují pomocí polynomu $A(x) = a_{r-1}.x^{r-1} + ... + a_1.x^1 + a_0.x^0$. V tomto kontextu jsou prvky tělesa všechny polynomy jedné proměnné stupně menšího než r s koeficienty z prvků tělesa $\mathbb{Z}_p$.
Stupeň polynomu A se značí deg$A$.

Těleso je dále určeno *generujícím polynomem* $G(x)$ (též definičním, vytvářecím, jinak značeno také $p(x)$, nebo $\pi(x)$), jenž musí být stupně r – o stupeň vyšší než polynomy tělesa a zároveň nimi nerozložitelný (ireducibilní).

Operace sčítání prvků tělesa je definována jako $A(x) + B(x) = C(x) : c_i = a_i + b_i$, kde součet koeficientů probíhá v aritmetice mod p. Stejným způsobem funguje rozdíl.

Operace násobení funguje jako konvenční součin polynomů s rozdílem, že součet koeficientů opět probíhá v aritmetice mod p. Výsledkem tohoto součinu ale může být polynom stupně $2(r-1)$. Ten ale není obsažen mezi prvky tělesa, ty mají stupeň nejvýše $r-1$. Takovýto výsledek by tedy porušoval jeho uzavřenost, proto je v takovém případě výsledkem zbytek po dělení tohoto součinu definičním polynomem. Symbolicky zapsané: $A(x) \cdot B(x) \bmod G(x)$. Dělitel (definiční polynom) je rozšířen na stupeň polynomu výsledku součinu a je od něj odečten, krok se opakuje s mezivýsledkem, dokud se mezivýsledek nestane platným prvkem tělesa a to je výsledek operace součinu tělesa.

Definice 2.2.4. *Primitivní polynom* značený obvykle α dokáže svými mocninami vyjádřit všechny nenulové prvky tělesa, čili $GF(n) = \{\alpha^i; i = 0..n-2\} \bigcup \{0\}$. V každém konečném tělese je alespoň jeden. ([5], kap. 4. §2.)

Toto poskytuje další způsob vyjádření prvků tělesa a to v exponenciálním tvaru: 0, α^0, α^1, α^2,...[4]
Protože umocnění je uzavřenou operací, pro mocniny větší než $n-2$ dochází k opakování polynomů. Platí:

$$\alpha^i = \alpha^{i \bmod (n-1)}, \text{ a tedy i: } \alpha^c.\alpha^d = \alpha^{(c+d) \bmod (n-1)}$$

[3] Označení konečného tělesa ($GF(n)$, $\mathbb{Z}_p$,...) také může zjednodušeně označovat pouze množinu prvků daného tělesa, což poznáme z kontextu. Stejně jako se to tak používá v mnohé literatuře, tak je to tímto způsobem užito i zde.

[4] V některé literatuře bývá 0 zapisována jako $\alpha^{-\infty}$.

Tento způsob násobení dvou prvků tělesa se nazývá logaritmický a umožňuje provádět i dělení.

V zabezpečovacích kódech nad $GF(p^r)$ odpovídá p kardinalitě abecedy a r počtu zabezpečovacích prvků zprávy. Pro praktické využití v digitálních systémech jsou nejvýznamnější tělesa $GF(2^r)$, která nás jediná budou zajímat dále. Prvek tohoto tělesa zapisujeme taky dvojkově jako $a_{r-1}...a_1\ a_0$, např. 1101 odpovídá polynomu $x^3 + x^2 + 1$ a dá se zapsat i dekadicky jako 13. V aritmetice mod 2 platí navíc další užitečné vlastnosti pro operaci s prvky tělesa. Každý prvek tělesa je sám sobě inverzním vzhledem ke sčítání $a \pm a = 0$. Díky tomu lze odvodit pravidlo pro umocňování na druhou $(a + b)^2 = a^2 + ab + ab + b^2 = a^2 + b^2$. Z toho plyne i: $(a + b + c)^2 = ((a + b) + c)^2 = a^2 + b^2 + c^2$, $(a + b)^{2q} = a^{2q} + b^{2q}; q \in \mathbb{N}$.

Následující příklady předvádí počítání s prvky konečných těles a utvoření konečného tělesa.

Příklad 2.2.1. Součet prvků $C(x) = A(x) + B(x) = ?$, kde $A(x) = x^3 + x^2 + 1$ a $B(x) = x^3 + x$. $C(x) = x^3 + x^3 + x^2 + x + 1 = x^2 + x + 1$. Další způsoby zápisu binárně: $1101 + 1010 = 0111$ a dekadicky: $13 + 10 = 7$. Je zřejmé, že pro operaci součtu postačuje znát charakteristiku tělesa. Stupeň rozšíření, ani generátor tělesa není tedy potřeba.

Příklad 2.2.2. Násobení prvků z předchozího příkladu s daným $G(x) = x^4 + x + 1$, z něhož vyplývá, že se jedná o prvky z tělesa $GF(16)$. Násobení se skládá ze dvou kroků, a to samotného konvenčního násobení a části modulo. Část násobení:

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

Tento výsledek je nutno dále vydělit generujícím polynomem, kde zbytek po dělení je finálním výsledkem operace násobení. Definiční polynom se vynásobí mocninou x, tak aby nabýval stupně mezivýsledku a od mezivýsledku se odečte. Tento postup se opakuje, dokud není polynom stupně menšího než 4. V tomto případě vypadá postup takto:

$$\begin{array}{r} (x^6 + x^5 + x^4 + x)/(x^4 + x + 1) = x^2 + x + 1 \\ \underline{-x^6 - x^3 - x^2} \\ x^5 + x^4 + x^3 + x^2 + x \\ \underline{-x^5 - x^2 - x} \\ x^4 + x^3 \\ \underline{-x^4 - x - 1} = \underline{\underline{x^3 + x + 1}} \end{array}$$

Tedy $A(x).B(x) = (x^3 + x^2 + 1).(x^3 + x) = x^3 + x + 1$.

Příklad 2.2.3. *Konstrukce GF(16).* Těleso je složeno z prvků (a_0, a_1, a_2, a_3). Pro ustanovení tělesa je potřeba najít ireducibilní polynom stupně 4. Celkově je 16 polynomů tohoto stupně, nás však nezajímají ty s nulovým absolutním členem, jelikož jsou triviálně rozložitelné jako $x.A(x)$. Ireducibilní najdeme tak, že vyřadíme ty, které vzniknou součinem: $\{A.B; A, B \in GF(16) \wedge \deg A + \deg B = 4 \wedge a_0, b_0 \neq 0\}$. To jsou dvojice A stupně 3 a B 1, nebo A i B stupně 2 – dohromady 4 varianty:

$$\begin{aligned} (x^3 + 1).(x + 1) &= x^4 + x^3 + x + 1 \\ (x^3 + x + 1).(x + 1) &= x^4 + x^2 + 1 \\ (x^3 + x^2 + 1).(x + 1) &= x^4 + x^3 + x^2 + x + 1 \\ (x^3 + x^2 + x + 1).(x + 1) &= x^4 + 1 \\ (x^2 + 1)^2 &= x^4 + 1 \\ (x^2 + x + 1).(x^2 + 1) &= x^4 + x^3 + x + 1 \\ (x^2 + x + 1)^2 &= x^4 + x^2 + 1 \end{aligned}$$

Možné definiční polynomy tedy jsou: x^4+x^3+1, x^4+x+1, $x^4+x^3+x^2+1$ a x^4+x^2+x+1.

Příklad 2.2.4. Sestavení logaritmické tabulky tělesa $GF(16)$ s $G(x)=x^4+x+1$, abychom mohli počítat operace násobení a dělení v této formě. Jako primitivní polynom zvolíme kupř. $\alpha=x$, pro který platí vzhledem k definičnímu polynomu $\alpha^4=x+1$. A podle toho sestavíme tabulku 2.3 prvků tělesa.

Indexová forma α^i	Polynomiální vyjádření (zbytek po dělení $G(x)$)	Dekadické vyjádření
–	0	0
α^0	1	1
α^1	x	2
α^2	x^2	4
α^3	x^3	8
α^4	$x+1$	3
α^5	x^2+x	6
α^6	x^3+x^2	12
α^7	x^3+x+1	11
α^8	x^2+1	5
α^9	x^3+x	10
α^{10}	x^2+x+1	7
α^{11}	x^3+x^2+x	14
α^{12}	x^3+x^2+x+1	15
α^{13}	x^3+x^2+1	13
α^{14}	x^3+1	9
$\alpha^{15}=\alpha^0$	1	1

Tabulka 2.3: Logaritmická tabulka prvků tělesa $GF(16)$ s $G(x)=x^4+x+1$

Příklad 2.2.5. Součin a podíl polynomů z příkladu 2.2.2 v logaritmické formě pomocí tabulky 2.3. Operandy převedeme vyhledáním v tabulce na mocniny primitivního polynomu, s kterými počítáme dále:

$$(x^3+x^2+1).(x^3+x)=\alpha^{13}.\alpha^9=\alpha^{21}=\alpha^7=x^3+x+1$$

Což odpovídá výsledku z předcházejícího příkladu.
Podíl je obdobný:

$$(x^3+x^2+1)/(x^3+x)=\alpha^{13}/\alpha^9=\alpha^{13-9}=\alpha^4=x+1$$

Využít jde i postupu převedení na součin za využití inverzního prvku dle definice 2.2.1. Inverzní prvek k α^9 je $(\alpha^9)^{-1}=\alpha^6$ (platí $\alpha^9\alpha^6=1$) a tedy podíl lze vypočíst jako součin: $(x^3+x^2+1).(x^3+x^2)=x+1$.

2.2.2 Lineární blokové kódy

Označili jsme-li v předchozí kapitole (systematické) blokové kódy jako nejvýznačnější, pak lineární blokové kódy jsou nejrozšířenější skupinou těchto kódů. Ne každý kód tohoto typu je systematický,

jak si ukážeme později. Pro připomenutí: systematičnost kódu znamená, že každý prvek zprávy má buď informační, nebo kontrolní charakter; blokovost kódu, že je vysílaná zpráva zabezpečena vždy po blocích pevně dané velikosti určené konkrétním kódem. Čerpáno z ([7], kap. 2.5–6 a 3) a ([2], kap. 3–3.2 a 4.3).

Definice 2.2.5. ([7], kap. 3.1) *Blokový kód* délky n, velikosti p^k s p symboly je množina p^k kódových slov (p-árních vektorů – prvek vektoru je jedním z p symbolů) délky $n > k$.

Definice *lineárního blokového kódu* je nejčastěji určena pomocí *utvářecí matice* **G** (také zabezpečovací, angl. generator matrix) a zároveň se tímto způsobem vždy dá vyjádřit. Matice pro kód (n, k) má velikost $k \times n$ a jedná se o matici nad tělesem $\mathbb{Z}_p$ pro kardinalitu abecedy p. To znamená, že její prvky $g_{i,j}$ jsou z tohoto tělesa. Má tento tvar:

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \\ \vdots \\ \mathbf{G}_k \end{bmatrix} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{bmatrix}$$

Matice je využita pro operaci zakódování. Vstupní blok zabezpečované zprávy délky k je vyjádřen *nezabezpečeným vektorem* $\mathbf{U} = (u_1, u_2, \ldots u_k)$, výstupní blok délky n poté *zabezpečeným vektorem* $\mathbf{V} = (v_1, v_2, \ldots v_n)$. Tato dvojice vektorů se jindy nazývá *informační vektor* a *zakódovaný vektor*. Proces zakódování probíhá jako násobení matic, pochopitelně v aritmetice mod p:

$$\mathbf{V} = \mathbf{U} \times \mathbf{G}$$

$$v_i = \sum_{j=1}^{k} u_j.g_{i,j} \text{ pro } i = 1, ..., n$$

Celou zprávu zabezpečíme tak, že ji rozdělíme na bloky (informační vektory), na které jednotlivě zakódování maticí aplikujeme.

Řádky $\mathbf{G}_i$ utvářecí matice jsou ve skutečnosti některými kódovými slovy daného kódu (*utvářecí slova*), pro které musí platit, že jsou lineárně nezávislé. Zbylé kódové slova vzniknou jako lineární kombinace těchto slov. Žádný z řádků matice tedy nesmí být možné vyjádřit jako součet násobků zbylých řádků matice. Pokud by tomu tak bylo, některým různým informačním vektorům by byli vypočteny stejné zabezpečené vektory – nejednalo by se o kódovací operaci, neboť by byla porušena obousměrnost převodu. Zakódování tak můžeme popsat rovněž jako lineární kombinaci utvářecích slov:

$$\mathbf{V} = u_1.\mathbf{G}_1 + u_2.\mathbf{G}_2 + ... + u_k.\mathbf{G}_k$$

Důsledkem je, že kombinace (součet) 2 nebo více kódových slov je opět kódové slovo. To je důvodem, proč se tyto kódy nazývají lineární. Toto se dá vyjádřit formálněji pomocí vektorových prostorů (dále viz definice 2.2.7), k tomu je ale potřeba vysvětlit další pojmy.

Vektorový prostor nad tělesem F o k dimenzích je množina vektorů V o k složkách s hodnotami z F, nad kterými je definována operace sčítání $+$ tvořící komutativní grupu. Dále operace násobení $\cdot$ kombinující prvek z F a V, kde výsledkem je prvek z V. Násobení je asociativní a distributivní. Skupina k lineárně nezávislých vektorů vytvářící svými kombinacemi celý prostor V se nazývají *báze* prostoru.

Definice 2.2.6. Nechť S je podmnožina vektorového prostoru V nad F. Nazveme ji *vektorový podprostor* prostoru V, pokud platí, že S s operacemi $+, \cdot$ tvoří opět vektorový prostor nad F a ten je algebraicky uzavřený:

1. $\forall \mathbf{X}, \mathbf{Y} \in S : \mathbf{X} + \mathbf{Y} \in S$

2. $\forall \mathbf{X} \in S, \forall a \in F : a \cdot \mathbf{X} \in S$

Definice 2.2.7. Blokový kód délky n a velikosti p^k je *lineární kód* právě tehdy, tvoří-li množina všech jeho kódových slov k-dimenzionální vektorový podprostor prostoru tvořeného všemi p-árními vektory délky n (tj. kódovými i nekódovými slovy).

A tedy utvářecí slova $\mathbf{G}_i$ jsou báze vektorového prostoru kódových slov.

Lin. blokové kódy, jež jsou systematické se obvykle tvoří tím způsobem, že zabezpečený blok tvoří *informační úsek*, což je původní nezabezpečený blok (vektor $\mathbf{V}_I$ obsahující všech k informačních symbolů) a za něj se připojí dopočítaný *kontrolní úsek* (vektor $\mathbf{V}_C$ obsahující r kontrolních symbolů). Utvářecí matici $\mathbf{G}_C$ takovéhoto kódu lze rozdělit na *informační podmatici* $\mathbf{I}$ velikosti $k \times k$ a *zabezpečovací podmatici* $\mathbf{C}$ velikosti $k \times r$:

$$\mathbf{G}_C = [\mathbf{I}; \mathbf{C}]$$

Informační matice je jednotková matice, pouze tedy kopíruje hodnoty ze vstupního vektoru do výstupního. O takové utvářecí matici se hovoří, že má *kanonický tvar* (angl. canonical form, nebo Standard Echelon Form – SEF). Zakódování v tomto případě můžeme zapsat jako:

$$\mathbf{V} = \mathbf{V}_I \cdot \mathbf{V}_C = \mathbf{U} \times \mathbf{I} \cdot \mathbf{U} \times \mathbf{C} = (u_1, ..., u_k) \cdot (v_{k+1}, ..., v_n) = (u_1, ...u_k, v_{k+1}, ...v_n)$$

kde operace $\cdot$ nad vektory značí jejich konkatenaci. Každá utvářecí matice systematického lineárního kódu lze převést do kanonického tvaru základními maticovými úpravami.

Systematičnost kódu umožňuje příjemci ověřit bezchybnost přijatého bloku (vektoru) dat $\mathbf{R}$ tím, že pro něj znovu vypočte kontrolní symboly a porovná je, zda jsou stejné s těmi přijatými. Pro kanonický tvar by kontrola správnosti vypadala následovně:

$$\mathbf{R}_I \times \mathbf{C} = \mathbf{R}_C$$

Příklad 2.2.6. Pomocí binárního lin. blokového kódu $(4, 3)$ daného maticí $\mathbf{G} = \left| \begin{smallmatrix} 1 & & & 1 \\ & 1 & & 1 \\ & & 1 & 1 \end{smallmatrix} \right|$[5] zabezpečte zprávu 110.
Protože se jedná o binární kód, je kódování počítáno v aritmetice mod 2.

$$\mathbf{V} = \mathbf{U} \times \mathbf{G} = (1, 1, 0) \times \mathbf{G} = (u_1, u_1 + u_2, u_2 + u_3, u_3) = (1, 0, 1, 0) \qquad \text{nebo}$$

$$\mathbf{V} = 1.\mathbf{G}_1 + 1.\mathbf{G}_2 + 0.\mathbf{G}_3 = (1, 1, 0, 0) + (0, 1, 1, 0) = (1, 0, 1, 0)$$

Výsledná zabezpečená zpráva je tedy 1010. Kód není systematický, protože informační symbol u_2 není přenášen explicitně.

Příklad 2.2.7. Kód z předchozího příkladu pozměňte do systematického tvaru převedením utvářecí matice $\mathbf{G}$ do kanonického tvaru $\mathbf{G}'$. Zakódujte pomocí něj totožnou zprávu 110.

Kanonická matice bude vypadat takto $\left| \begin{smallmatrix} 1 & & & ? \\ & 1 & & ? \\ & & 1 & ? \end{smallmatrix} \right|$ z čehož vidíme, že třetí řádek není třeba měnit $\mathbf{G}'_3 = \mathbf{G}_3$. Ostatní 2 získáme kombinací původních: $\mathbf{G}'_1 = \mathbf{G}_1 + \mathbf{G}_2 + \mathbf{G}_3$ a $\mathbf{G}'_2 = \mathbf{G}_2 + \mathbf{G}_3$. Tedy:

$$\mathbf{G}' = \begin{bmatrix} 1 & & & 1 \\ & 1 & & 1 \\ & & 1 & 1 \end{bmatrix}$$

$$\mathbf{V}' = \mathbf{U} \times \mathbf{G}' = (1, 1, 0) \times \mathbf{G}' = (u_1, u_2, u_3, u_1 + u_2 + u_3) = (1, 1, 0, 0)$$

Zabezpečená zpráva pozměněným kódem je $110\underline{0}$ – podtržen je kontrolní úsek bloku. Všimněte si, že transformací jsme získali sudou paritu. Kód má stejnou náročnost (stejný počet výpočetních elementů – součet jedničkových polí v matici) jako ten původní.

[5] Vynechaná místa značí konstantu 0.

Dosud jsme se zabývali stranou odesílatele, nyní přejděme ke straně příjemce. Každý lin. blokový kód má svou *kontrolní matici* $\mathbf{H}$ velikosti $r \times n$ tvaru:

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \dots & h_{1,n} \\ h_{2,1} & h_{2,2} & \dots & h_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{r,1} & h_{r,2} & \dots & h_{r,n} \end{bmatrix} \quad \text{pro kterou platí} \quad \begin{aligned} \mathbf{G} \times \mathbf{H}^T &= \mathbf{0} \\ \mathbf{U} \times \mathbf{G} \times \mathbf{H}^T &= \mathbf{0} \\ \mathbf{V} \times \mathbf{H}^T &= \mathbf{0} \end{aligned}$$

Její transponovanou verzi příjemce použije na ověření platnosti přijaté zprávy a získává tak *chybový syndrom* (vektor syndromu) $\mathbf{S}$:

$$\mathbf{R} \times \mathbf{H}^T = \mathbf{S} = (s_1, ..., s_r)$$

jenž je roven nulovému vektoru, pakliže k chybě při přenosu nedošlo. A pokud nedošlo, platí $\mathbf{V} = \mathbf{R}$. V kanonickém tvaru lze také kontrolní matici rozdělit na dvě podmatice:

$$\mathbf{H}_C = [\mathbf{C}^T; \mathbf{1}] \quad \text{resp.} \quad \mathbf{H}_C^T = \begin{bmatrix} \mathbf{C} \\ \mathbf{1} \end{bmatrix}$$

kde $\mathbf{C}$ je zabezpečovací podmatice z utvářecí matice a $\mathbf{1}$ značí jednotkovou matici.

Příklad 2.2.8. Sestavte kontrolní matici ke kódu z předchozího příkladu zadaného maticí $\mathbf{G}'$. Pomocí ní vypočtěte vektor syndromu a ověřte tak platnost přijatého vektoru dat $\mathbf{R} = (1, 1, 1, 1)$ a téhož vektoru s chybou při přenosu na 3. symbolu zprávy, tj. $\mathbf{R}' = (1, 1, 0, 1)$, a vektoru se shlukovou dvojchybou $\mathbf{R}'' = (1, 1, 0, 0)$.
Kontrolní matice má jediný řádek odpovídající jedinému kontrolnímu symbolu a je rovna:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Transponovanou ji použijeme na výpočet syndromů:

$$(1,1,1,1) \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = [0] \qquad (1,1,0,1) \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = [1] \qquad (1,1,0,0) \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = [0]$$

$\mathbf{R}$ je vyhodnocen jako platný, $\mathbf{R}'$ správně jako chybný, nicméně $\mathbf{R}''$ špatně jako platný, jelikož sudá parita je schopna detekovat pouze lichý počet chyb, sudý počet chyb totiž změní kódové slovo v jiné kódové slovo.

Situace, v kterých je odeslaný vektor $\mathbf{V}$ odlišný od přijatého vektoru $\mathbf{R}$ v důsledku ovlivnění šumem zapisujeme jako $\mathbf{R} = \mathbf{V} + \mathbf{E}$, kde $\mathbf{E}$ je tzv. *chybový vektor*. Objevenou chybu se tedy snažíme opravit tak, že hledáme $\mathbf{E}$, o jehož hodnoty provedeme korekci přijatého vektoru dat $\mathbf{V} = \mathbf{R} + (-\mathbf{E})$, resp. u binárních a jiných 2^p-árních kódů $\mathbf{V} = \mathbf{R} + \mathbf{E}$. Syndrom chyby závisí pouze na vektoru chyby:

$$\mathbf{S} = \mathbf{R} \times \mathbf{H}^T = (\mathbf{V} + \mathbf{E}) \times \mathbf{H}^T = \mathbf{V} \times \mathbf{H}^T + \mathbf{E} \times \mathbf{H}^T = \mathbf{E} \times \mathbf{H}^T$$

Hledání chybového vektoru pro konkrétní syndrom je možno pomocí velkého množství heuristik – cílem je přiřazení nejpravděpodobnější chyby danému syndromu v omezeném čase. Popis těchto heuristik pro obecné lin. blokové kódy by bylo nad rámec této práce.

Důležitou charakteristikou kódu je jeho detekční schopnost, zejména kódová vzdálenost. Tu dokážeme u lineárních kódů zjistit poměrně snadno pomocí tzv. (Hammingových) vah.

Definice 2.2.8. *Hammingova váha* vektoru $\mathbf{V}$ značená $w\{\mathbf{V}\}$ udává počet jeho nenulových složek.

Jinými slovy je to Hammingova vzdálenost od nulového vektoru $w\{\mathbf{V}\} = d(\mathbf{V}, \mathbf{0})$. A kódová vzdálenost je u konkrétního kódu rovna nejmenší z vah všech nenulových vektorů prostoru V kódových slov, značeno $d_{min} = w_{min}(V - \mathbf{0})$. Tj. váha kódového slova s nejmenším počtem nenulových symbolů (u binárního kódu jedničkových), avšak alespoň s jedním. To platí díky tomu, že existují-li 2 různé vektory $\mathbf{X}$ a $\mathbf{X}'$ s vzdáleností d, existuje také odlišný vektor $\mathbf{Y}$ s vahou $w = d$. Což se dá snadno dokázat:

Totiž existuje-li tato dvojice kódových vektorů a $d(\mathbf{X}, \mathbf{X}') = 1$ – liší se v jednom, i. symbolu $(1 \leq i \leq n)$, tedy $\mathbf{X} = (x_1, ..., x_i, ..., x_n)$ a $\mathbf{X}' = (x_1, ..., x_i', ..., x_n)$.
Potom výsledkem jejich kombinace je vektor $\mathbf{Y} = \mathbf{X} + (-\mathbf{X}') = (0, ..., y_i, ..., 0)$ s $w\{\mathbf{Y}\} = d = 1$. Podobně se dá postupovat s dvojicemi lišícími se ve 2 až n symbolech. Q.E.D.

Protože v této práci budeme pracovat i s vyšším kódem než jen binárním (tj. $p > 2$), následuje ukázka takového lin. blokového kódu. Pro výpočty je třeba si uvědomit, že jednotlivé prvky vektorů i matic jsou z tělesa $GF(p)$, tedy i operace mezi nimi probíhají v aritmetice tohoto tělesa.

Příklad 2.2.9. Zakódujte informační vektor $\mathbf{U} = (3, 8, 14, 2)$ sedenárním kódem $(6, 4)$ nad $GF(16)$[6] s $G(x) = x^4 + x + 1$ specifikovaným utvářecí maticí níže. K zabezpečenému vektoru přičtěte chybový vektor $\mathbf{E} = (4, 0, 7, 0, 0, 0)$ a spočtete jeho syndrom $\mathbf{S}$.

$$\mathbf{G} = \begin{bmatrix} 1 & & & & 1 & 9 \\ & 1 & & & 3 & 11 \\ & & 1 & & 5 & 13 \\ & & & 1 & 7 & 15 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & 3 & 5 & 7 & 1 & \\ 9 & 11 & 13 & 15 & & 1 \end{bmatrix}$$

Provedeme zabezpečení informačního vektoru. Násobení je možno provádět logaritmicky pomocí tabulky 2.3.

$$\mathbf{V} = \mathbf{U} \times \mathbf{G} = (3, 8, 14, 2, 5, 11)$$

$$\begin{aligned} v_5 &= 1.3 + 3.8 + 5.14 + 7.2 = \\ &= 3 + \alpha^{4+3} + \alpha^{8+11} + \alpha^{10+1} = \\ &= 3 + 11 + 3 + 14 = \\ &= 5 \end{aligned} \qquad \begin{aligned} v_6 &= 9.3 + 11.8 + 13.14 + 15.2 = \\ &= \alpha^{14+4} + \alpha^{7+3} + \alpha^{13+11} + \alpha^{12+1} = \\ &= 8 + 7 + 9 + 13 = \\ &= 11 \end{aligned}$$

Nyní získáme přijatý vektor připočtením chyby k odeslanému:

$$\mathbf{R} = \mathbf{V} + \mathbf{E} = (3, 8, 14, 2, 5, 11) + (4, 0, 7, 0, 0, 0) = (7, 8, 9, 2, 5, 11)$$

A nakonec spočteme syndrom:

$$\mathbf{S} = \mathbf{R} \times \mathbf{H}^T = (5, 10)$$

$$s_1 = 1.7 + 3.8 + 5.9 + 7.2 + 1.5 = 7 + 11 + 12 + 14 + 5 = 5$$

$$s_2 = 9.7 + 11.8 + 13.9 + 15.2 + 1.11 = 5 + 7 + 14 + 13 + 11 = 10$$

[6] Číselné prvky vektoru a matice jsou decimálním zápisem prvku tělesa $GF(16)$.

Hammingův kód

Optimální třídu binárních systematických lin. blokových kódů objevil R. W. Hamming v roce 1950. Tyto kódy byly používány pro dálkovou telefonii. Jedná se o korekční kódy se vzdáleností $d_{min} = 3$, řazené tedy mezi SEC-DED kódy schopné buď detekovat 2 chyby, nebo 1 chybu detekovat i opravit. Nazvané jsou po objeviteli *Hammingovy kódy* a značeny $HK(n, k)$. Pro r kontrolních symbolů má zabezpečený blok velikost $n = 2^r - 1$ a obsahuje $k = 2^r - 1 - r = n - r$ informačních symbolů. Jejich optimálnost spočívá právě v tom, že každá varianta má právě tolik kontrolních symbolů, kolik jich je teoreticky minimálně vyžadováno pro dosažení této kódové vzdálenosti na blok jeho velikosti. Tím dosahuje jako kód nejnižší možné míry redundance pro tuto d_{min}. Skvěle se tak hodí pro přenosy s ojedinělými náhodnými chybami.

Předvedeme si výstavbu obecného Hammingova kódu v kanonickém tvaru. Nejprve přečíslujeme indexy symbolů informační části výstupního vektoru (a také symboly vstupního vektoru) z lineárního indexování $v_1, v_2, v_3, ..., v_k$ na nelineární $v_3, v_5, ..., v_7, v_9, ..., v_n$. To pouze vynecháme indexy, jež jsou mocninou čísla 2. Těch je právě r, proto poslední symbol má nově index $k + r = n$. Pro zbylých r kontrolních symbolů výstupního vektoru naopak použijeme nevyužité indexy $v_1, v_2, ..., v_{2^{(r-1)}}$. Zabezpečovací podmatice **C** se potom skládá z řádků, obsahujících binární zápis (nejméně významný bit je vlevo) indexu informačního symbolu k němu příslušícímu. Příjemcem vypočteným syndrom **S** potom v případě chyby obsahuje binární zápis indexu, na kterém symbolu došlo k chybě.

Postup pro konkrétní kód $HK(7, 4)$ je následovný. Přečíslujeme vstupní a výstupní vektory:

$$\mathbf{U} = (u_3, u_5, u_6, u_7) \qquad \mathbf{V} = (v_3, v_5, v_6, v_7, v_1, v_2, v_4)$$

Sestavíme vytvářecí matici a podle ní kontrolní matici postupem pro kanonický tvar:

$$\mathbf{G} = \begin{array}{c} \\ u_3 \\ u_5 \\ u_6 \\ u_7 \end{array} \begin{array}{c} \begin{array}{cccc} v_3 & v_5 & v_6 & v_7 \end{array} \\ \left[\begin{array}{cccc|ccc} 1 & & & & 1 & 1 & \\ & 1 & & & 1 & & 1 \\ & & 1 & & & 1 & 1 \\ & & & 1 & 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_4 \end{array}$$

$$\mathbf{H} = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{cccc} v_3 & v_5 & v_6 & v_7 \end{array} \\ \left[\begin{array}{cccc|ccc} 1 & 1 & & 1 & 1 & & \\ 1 & & 1 & 1 & & 1 & \\ & 1 & 1 & 1 & & & 1 \end{array}\right] \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_4 \end{array}$$

2.2.3 Cyklické kódy

Významnou podskupinou lineárních blokových kódů jsou cyklické kódy. Nezadávají se utvářecí maticí jako ostatní kódy tohoto druhu, nýbrž tzv. *vytvářecím mnohočlenem* $G(x)$. Tento způsob zápisu je efektivnější – cyklický kód (n, k) lze zadat nerozložitelným mnohočlenem stupně r namísto matice velikosti $k \times r$:

$$g(x) = g_r.x^r + ... + g_1.x^1 + g_0$$

Tato kapitola čerpá z ([7], kap. 4–4.4).

Definice 2.2.9. ([7], kap. 4.2) *Cyklický kód* je takový lineární blokový kód, pro něhož platí, že cyklickým posunem složek jeho kódového vektoru vzniká jiný kódový vektor.

Máme-li kódový vektor $\mathbf{V} = (v_1, v_2, ..., v_{n-j}, v_{n-j+1}, ..., v_n)$ takového kódu, jeho cyklickým posunutím o $j < n$ pozic doprava vzniká kódový vektor $\mathbf{V'} = (v_{n-j+1}, ..., v_n) \cdot (v_1, v_2, ..., v_{n-j})$.

Cyklické kódy existují v nesystematické i systematické formě. Kódový vektor **V** vyjádříme jako mnohočlen $v(x) = v_n.x^{n-1} + ... + v_2.x^1 + v_1$. Podobně i informační vektor **U** a jakýkoliv jiný datový vektor šířky m zavedený v předcházející kapitole můžeme vyjádřit jako mnohočlen stupně

$m-1$. Poté můžeme zakódování zapsat jako $v(x) = u(x).g(x)$ pro nesystematický cyklický kód. Roznásobením získáme maticový tvar (je možno všimnout si kýženého cyklického opakování):

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & \cdots & g_r & & \\ & g_0 & g_1 & \cdots & g_r & \\ & & \ddots & & & \ddots \\ & & & g_0 & g_1 & \cdots & g_r \end{bmatrix}$$

Vidíme, že korektně přijaté kódové slovo musí být dělitelné $G(x)$ beze zbytku a získáme jej právě vydělením tímto polynomem:

$$\frac{r(x)}{g(x)} = u(x) \quad \text{pokud} \quad r(x) = v(x) \qquad (2.1a) \qquad\qquad \frac{r(x)}{g(x)} = a'(x) + \frac{s(x)}{g(x)} \quad \text{jinak} \qquad (2.1b)$$

Pokud není, dorazila chybná zpráva a poznáme to nenulovým zbytkem $s(x) = r(x) \bmod g(x)$, tedy nenulovým syndromem chyby.

Pokud $x^r.u(x)$ vydělíme $g(x)$, zjistíme tuto rovnost $x^r.u(x) = a(x).g(x) + b(x)$, kde $a(x)$ je kvocient a $b(x)$ zbytek. Jak jsme si uvedli výše, $a(x).g(x)$ představuje kódové slovo, proto je kódové slovo i $x^r.u(x) - b(x) = a(x).g(x)$. Protože $b(x)$ je stupně menšího než r, jedná se o systematický způsob zakódování:

$$v(x) = x^r.u(x) - (x^r.u(x) \bmod g(x))$$

Ačkoliv je to modifikace kódu, prostor kódových slov zůstává stejný. To znamená, že je zachována cyklická vlastnost kódu a dělitelnost beze zbytku každého kódového slova. Tedy i způsob detekce chyby může zůstat stejný. Tímto způsobem jsme vždy schopni převádět mezi sebou systematický a nesystematický cyklický kód. Postup získání zbytku po dělení je pomocí běžného tzv. *dlouhého dělení* mnohočlenů.

Příklad 2.2.10. Zakódujte informační mnohočlen $u(x) = x^3 + x^2 + x$ pomocí vytvářecího mnohočlenu $g(x) = x^3 + x + 1$ binárního cyklického kódu $(7, 4)$ nesystematicky do $v(x)$ a systematicky do $v'(x)$ (tzv. CRC-3 – viz dále).

Pro $v(x)$ sestavíme utvářecí matici:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & & 1 & & & \\ & 1 & 1 & & 1 & & \\ & & 1 & 1 & & 1 & \\ & & & 1 & 1 & & 1 \end{bmatrix}$$

Vypočteme
$\mathbf{V} = (0, 1, 1, 1) \times \mathbf{G} = (0, 1, 0, 0, 0, 1, 1)$,
čili $v(x) = x^6 + x^5 + x$.

Pro $v'(x)$ nejprve vytvoříme místo pro kontrolní symboly $x^r.u(x) = x^6 + x^5 + x^4$, poté zjistíme zbytek po dělení $g(x)$:

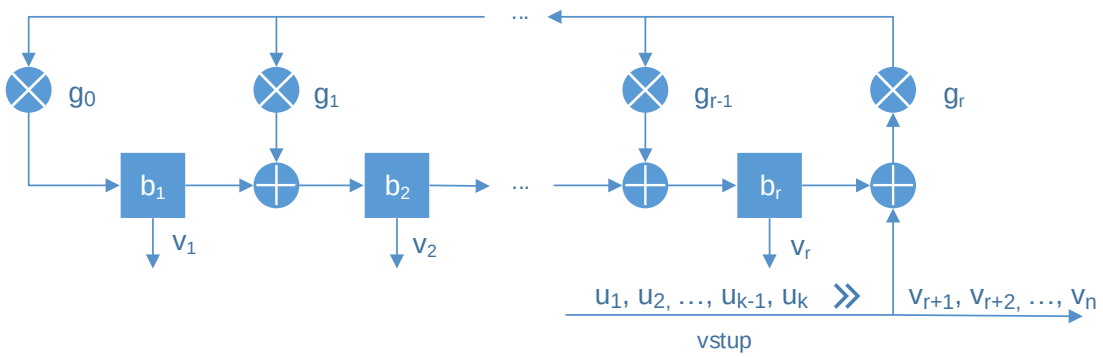
$$\begin{array}{r} (x^6 + x^5 + x^4)/(x^3 + x + 1) = x^3 + x + 1 \\ \underline{-x^6 - x^4 - x^3} \\ x^5 + x^3 \\ \underline{-x^5 - x^3 - x^2} \\ x^2 \end{array}$$

Vychází $v'(x) = x^6 + x^5 + x^4 + x^2$.

Systematické cyklické kódy se také označují zkratkou *CRC* (z angl. Cyclic Redudancy Check, cyklický redundantní součet). ([1], kap. 4.2) Tyto kódy ještě dělíme podle délky r zabezpečovací části kódu (resp. stupně r vytvářecího polynomu) na třídy CRC-r. Nejjednodušší je CRC-1 mající $g(x) = x + 1$ a je to kód sudé parity.

Cyklické kódy díky svému opakování umožňují efektivní sériovou implementaci pracující po 1 či více symbolech. Na obrázku 2.4 je obecné obvodové schéma CRC kodéru délky r nad tělesem

$GF(p)$. Blok sčítačky $\oplus$ a násobičky $\otimes$ provádějí operace $+$ a $\cdot$ s prvky tělesa. Na začátku jsou registry b_x inicializovány do výchozích hodnot a do obvodu se postupně přivádějí symboly z informačního vektoru. Po zpracování posledního symbolu obsahují registry zbytek $b(x)$. Funkce obvodu imituje kroky dlouhého násobení (jak bylo předvedeno v příkladu 2.2.10).



Obrázek 2.4: Sériový CRC-r kodér

2.2.4 Reed-Solomonovy kódy

Objeveny I. S. Reedem a G. Solomonem v roce 1960, patří *Reed-Solomonovy kódy* (RS) mezi nebinární systematické cyklické kódy. Je užitečné vědět, že patří pod třídu cyklických kódů BCH při speciální volbě parametrů. Na rozdíl od obecných cyklických kódů jsou BCH kódy samoopravné. RS kódy se pak staly všudypřítomné díky svojí vysoké zabezpečovací schopnosti, navíc uzpůsobené proti shlukovým chybám. A zároveň efektivní implementaci, protože vyšší míra algebraické struktury umožnila vzniku efektivním postupům opravy chyb. Používají se tak dnes např. v optických discích, digitálním televizním vysílání, DSL, v přenosech z vesmíru (sonda Voyager nebo Galileo) a samozřejmě v Ethernetu. Informace k podkapitole jsou čerpány z ([4], kap. 4–5), ([7], kap. 6).

RS kód značený $RS(n,k)$ je jako jiné cyklické kódy charakterizován vytvářecím mnohočlenem $g(x)$ stupně r, délkou zabezpečeného bloku k a tělesem symbolů $GF(p^m)$. Pro jeho vytvářecí mnohočlen platí, že má r kořenů. Jsou jimi po sobě jdoucí mocniny čísla α zvoleného z tělesa: $\alpha^{b+1}, \alpha^{b+2}, ..., \alpha^{b+r}$ začínající od zvoleného exponentu b – obvykle $b = 0$, nebo $b = -1$ ($\alpha^{b+1} = 1$). Vytvářecí mnohočlen se tak dá vyjádřit jako součin jeho r faktorů:

$$\begin{aligned} g(x) &= \prod_{i=1}^{r}(x - \alpha^{b+i}) = (x - \alpha^{b+1})(x - \alpha^{b+2})\dots(x - \alpha^{b+r}) \\ g(\alpha^j) &= 0 \quad \text{platí pro každý z faktorů, tj. } j = b+1, ..., b+r \\ g(x) &= x^r + g_{r-1}.x^{r-1} + \dots + g_1.x^1 + g_0 \end{aligned}$$

Na rozdíl od obecného cyklického kódu je zde koeficient g_r vždy 1. Podle toho, zda je α primitivní prvek tělesa se RS kód nazývá *primitivní*, nebo *neprimitivní*.

Je-li kód vystavěn nad tělesem $GF(p^m)$, tak zabezpečovaný blok má velikost $n \leq p^m - 1$. Tak lze každému symbolu kódového slova jednoznačně přiřadit prvek tělesa. Pro kódovou vzdálenost po zvolení k platí $d_{min} = r + 1$, a tedy jeho skutečná kódová vzdálenost odpovídá teoretické hodnotě. Jinými slovy neexistuje žádný jiný kód (n,k) s větší kódovou vzdáleností a RS je tedy *maximálně distančním kódem* (angl. maximum-distance code). Kód dokáže detekovat $r = 2t$ chyb bez opravy, nebo t chyb detekovat a opravit. Má-li zabezpečený blok velikost $n < p^m - 1$, jedná se *zkrácený RS kód*.

Operace zakódování probíhá totožně, jako bylo uvedeno v předchozí podkapitole. Operace dekódování za cílem detekce chyb může také proběhnout stejným způsobem, jaký byl uveden u cyklických kódů. Tedy ověřením dělitelnosti beze zbytku přijatého slova vytvářecím mnohočlenem podle rovnice 2.1b. Tímto způsobem detekujeme až r potencionálních chyb. Protože mnohočlen přijaté zprávy musí být dělitelný beze zbytku nejen samotným $g(x)$, ale i všemi jeho faktory, platí pro bezchybně přijatou zprávu:

$$S_i = r(x) \bmod x - \alpha^{b+i} = 0 \qquad \text{pro } i = 1, ..., r \tag{2.2}$$

kde S_i jsou *syndromy chyb* tvořící *chybový syndrom* $S(x) = S_r.x^r + \ldots + S_2.x^2 + S_1.x$ Zřejmě pro bezchybnou zprávu $S(x) = 0$. Polynom přijaté zprávy můžeme vyjádřit jako u jiných lineárních kódů součtem vyslaného polynomu a *chybového polynomu* (angl. error magnitudes polynomial) $r(x) = v(x) + e(x)$. Chybový polynom má v členů, kdy koeficient každého členu (y_x) představuje *velikost* (hodnotu) *chyby* a exponent (e_x) *pozici chyby* v přijaté zprávě. Předpokládáme však, že maximálně obsahuje $v \leq t$ členů, jinak by byla překročena zabezpečovací schopnost kódu.

$$e(x) = y_v.x^{e_v} + ... + y_2.x^{e_2} + y_1.x^{e_1}$$

Víme, že $v(x)$ je bezezbytku dělitelný $g(x)$, takže výpočet syndromů znamená vydělení samotného $e(x)$:

$$S_i = e(x) \bmod x - \alpha^{b+i}$$

a to sa často zapisuje, zejména v odborné literatuře, jako:

$$S_i = r(\alpha^{b+i}) = v(\alpha^{b+i}) + e(\alpha^{b+i}) = e(\alpha^{b+i}) \tag{2.3}$$

Pro přehlednost položíme $b = 0$ a dosadíme:

$$e(\alpha^i) = y_v.\left(\alpha^i\right)^{e_v} + ... + y_2.\left(\alpha^i\right)^{e_2} + y_1.\left(\alpha^i\right)^{e_1} \tag{2.4}$$

Výpočet jednoho syndromu tedy můžeme provést sekvenčním obvodem jako je na obrázku 2.4 dle rovnice 2.2 (v takovém případě nemá obvod kaskádu registrů, ale jediný registr), nebo paralelním obvodem dle rovnice 2.3. Oba přístupy vedou ke stejnému výsledku, totiž obecně platí pro libovolný polynom $o(x)$ stupně n:

$$o(x) \bmod x - \beta = o(\beta)$$

Což lze dokázat rozepsáním dlouhého násobení:

$$\begin{array}{l}
(o_n.x^n + o_{n-1}.x^{n-1} + ... + o_0)/(x-\beta) = o_n.x^{n-1} + (o_n.\beta + o_{n-1}).x^{n-2} + ... \\
\underline{-o_n.x^n + o_n.x^{n-1}.\beta} \\
\qquad (o_n.\beta + o_{n-1}).x^{n-1} + o_{n-2}.x^{n-2} + ... \\
\qquad \underline{-(o_n.\beta + o_{n-1}).x^{n-1} + (o_n.\beta + o_{n-1}).x^{n-2}.\beta} \\
\qquad\qquad (o_n.\beta^2 + o_{n-1}.\beta + o_{n-2}).x^{n-2} + ... \\
\qquad\qquad\qquad \ddots \\
\qquad\qquad\qquad\qquad o_n.\beta^n + o_{n-1}.\beta^{n-1} + ... + o_0 = o(\beta)
\end{array}$$

Dosazením všech faktorů do $e(x)$ (rovnice 2.4) získáme soustavu rovnic:

$$\begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_r \end{bmatrix} = \begin{bmatrix} (\alpha^{e_1}) & (\alpha^{e_2}) & \ldots & (\alpha^{e_v}) \\ (\alpha^{e_1})^2 & (\alpha^{e_2})^2 & \ldots & (\alpha^{e_v})^2 \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha^{e_1})^r & (\alpha^{e_2})^r & \ldots & (\alpha^{e_v})^r \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_v \end{bmatrix} \tag{2.5}$$

Řešením této soustavy můžeme zjistit velikost chyb, známe-li již pozice chyb a naopak.

Hledání pozic chyby převedeme na hledání kořenů *mnohočlenu chybového lokátoru* (angl. error-locator polynomial) tvaru:

$$\begin{aligned} \Lambda(x) &= (1 - x.\alpha^{e_1}).(1 - x.\alpha^{e_2}) \dots (1 - x.\alpha^{e_v}) \\ &= \Lambda_v.x^v + \Lambda_{v-1}.x^{v-1} + ... + \Lambda_1.x^1 + 1 \end{aligned} \tag{2.6}$$

Dále si zavedeme *mnohočlen vyčíslení chyb* (angl. error-evaluator polynomial):

$$\begin{aligned} \Omega(x) &= \Lambda(x).S(x) \bmod x^{r+1} \\ &= \Omega_t.x^r + \Omega_{r-1}.x^{r-1} + ... + \Omega_1.x \end{aligned} \tag{2.7}$$

Modulo zajistí po násobení ořezání všech členů stupně vyššího než r. Za povšimnutí stojí, že $\Omega_1 = S_1$. Z rovnice 2.7 vyplývá:

$$\frac{\Lambda(x).S(x)}{x^{r+1}} = Q(x) + \frac{\Omega(x)}{x^{r+1}}$$

kde polynom Q(x) obsahuje všechny členy stupně vyššího r ze součinu $\Lambda(x).S(x)$. Mezi mnohočlenem vyčíslení a lokátoru je samozřejmě vztah, který je zjevný po přeskládání poslední rovnice:

$$\Omega(x) = \Lambda(x).S(x) - Q(x).x^{r+1}$$

A sice, že $S(x)$ a x^{r+1} jsou soudělná, přičemž $\Omega(x)$ je jejich společný dělitel. To znamená, že jsme schopni pomoci *rozšířeného Euklidova algoritmu* pro hledání největšího společného dělitele (NSD) zjistit neznámé polynomy této rovnice. Máme-li dvě čísla různé od nuly a a b, existují takové dva koeficienty u a v pro něž platí:

$$NSD(a, b) = u.a - v.b$$

Pro náš případ a a b jsou $S(x)$ a x^{r+1}, u a v jsou $\Lambda(x)$ a $Q(x)$ a NSD je $\Omega(x)$. Následuje popis Euklidova algoritmu dle literatury ([7], kap. 5.5.2) zjednodušený pro výpočet pouze NSD a u.

Nastavíme proměnné $r_0 = a$, $r_1 = b$, $u_0 = 1$ a $u_1 = 0$. Pro krok $i > 1$ algoritmu provedeme dělení r_{i-2}/r_{i-1}, přičemž podíl uložíme do proměnné q a zbytek do proměnné r_i. Zároveň po vydělení vypočteme hodnotu $u_i = u_{i-2} - q.u_{i-1}$. Krok se opakuje ($i = i + 1$) dokud deg $r_i \geq t$. Výsledkem algoritmu je $NSD = r_{i-1}$ a $u = u_i$.

Skutečným výsledkem algoritmu jsou násobky hledaných hodnot, čili $u = c.\Lambda(x)$ a $NSD = c.\Omega(x)$. Protože však známe Ω_1, čili $c^{-1} = \frac{\Omega_1}{NSD_1}$, jsme schopni hledané hodnoty dopočítat. $\Omega_1 = S_1$ však nemusí být vždy nenulové, proto je výhodnější využít vlastnosti, že $\Lambda_1 = 1$, a tedy $c = u_1$.

Po zjištění chybového lokátoru je nutné najít jeho kořeny. To se děje pomocí hrubé síly – zkouší se dosadit všechny možnosti α^i pro $i = 0, 1, ..., p^m - 1$ do $\Lambda(x)$. Pro každý nalezený kořen spočteme inverzi a získáme tak pozici chyby $\alpha^{e_j} = \left(\alpha^i\right)^{-1}$. Vyzkoušením všech variant nejen, že zjistíme pozice chyb, ale známe již i hodnotu v určující počet chyb.

Po nalezení pozic chyb zbývá již jen vypočítat jejich hodnoty. To můžeme provést vyřešením v rovnic ze systému rovnic (2.5). Pro velká v je však efektivnější použít *Forneyho vzorec* (angl. Forney's equation) pro každou pozici chyby $j = 1, 2, ..., v$:

$$y_j = (\alpha^{e_j})^{1-b} . \frac{\Omega(\alpha^{-e_j})}{\Lambda'(\alpha^{-e_j})}$$

$\Lambda'(x)$ značí *formální derivaci* $\Lambda(x)$:

$$\begin{aligned}\Lambda'(x) &= v\Lambda_v.x^{v-1} + (v-1)\Lambda_{v-1}.x^{v-2} + ... + 2\Lambda_2.x^1 + \Lambda_1 \\ &= \sum_{i=1}^{v} \Lambda_v.x^{v-1} + \sum_{i=1}^{v-1} \Lambda_{v-1}.x^{v-2} + ... + \sum_{i=1}^{2} \Lambda_2.x^1 + \Lambda_1\end{aligned}$$

Koeficient, kterým se člen při derivaci násobí, je přirozené číslo, není to prvek z konečného tělesa, což lépe vyjadřuje druhý přepis rovnice. Nad $GF(2^m)$ tak každý člen se sudým exponentem při derivaci zmizí:

$$\Lambda'(x) = ... + 5\Lambda_5.x^4 + 3\Lambda_3.x^2 + \Lambda_1$$

Obecný postup v opravení chyby spočívá v postupu k nalezení chybového polynomu $e(x)$. Skládá se z těchto kroků:

1. výpočet chybového syndromu $S(x)$,
2. výpočet chybového lokátoru $\Lambda(x)$,
3. nalezení kořenů (pozic chyb) chybového lokátoru α^{-e_j},
4. výpočet velikostí chyb y_j,
5. sestavení chybového polynomu $e(x)$,
6. opravení přijaté zprávy odečtením chyby $v(x) = r(x) - e(x)$.

Existují další známé metody, které se v jednotlivých krocích dají použít. Asi nejznámější je Petersonův algoritmus pro výpočet $\Lambda(x)$, Berkelamp-Masseyův pro totéž a Chienovo prohledávání (angl. Chien search) pro nalezení kořenů.

Příklad 2.2.11. Najděte a opravte chybu v přijaté zprávě $r(x) = 14.x^7 + 5.x^6 + 2.x^5 + 13.x^4 + 8.x^3 + 9.x^2 + 4.x + 14$, jež vznikla zabezpečením $u(x) = 14.x^3 + 5.x^2 + 2.x + 14$ kódem $RS(8, 4)$ s $g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) = (x - 2)(x - 4)(x - 8)(x - 3)$ nad $GF(16)$ s $G(x) = x^4 + x + 1$.
Nejprve spočteme všechny syndromy (násobení můžeme provést logaritmicky pomocí tabulky 2.3):

$$S_2 = r(\alpha^2) = 14.9 + 5.15 + 2.7 + 13.5 + 8.12 + 9.3 + 4.4 + 14 = 12$$

Stejným způsobem spočteme ostatní syndromy:

$$S_1 = r(\alpha) = 1 \qquad S_3 = r(\alpha^3) = 14 \qquad S_4 = r(\alpha^4) = 3$$

A sestavíme mnohočlen chybového syndromu:

$$S(x) = 3.x^4 + 14.x^3 + 12.x^2 + x$$

Euklidovým algoritmem najdeme mnohočlen chybového lokátoru a zároveň vyčíslení chyby. Provedeme inicializaci:

$$r_0 = x^{r+1} = x^5 \qquad r_1 = S(x) \qquad u_0 = 1 \qquad u_1 = 0$$

Dělením r_0/r_1 získáme q_2 a r_2, spočteme u_2:

$$\begin{aligned}
(x^5)/(3.x^4{+}14.x^3+12.x^2+x) &= 14.x+8 = q_2\\
\underline{-x^5-11.x^4-\ 4.x^3-14.x^2} & \\
11.x^4+\ 4.x^3+14.x^2 & \\
\underline{-11.x^4-\ 9.x^3-10.x^2-8.x} & \\
13.x^3+4.x^2+8.x &= r_2
\end{aligned}$$

$$u_2 = u_0 + u_1.q_1 = 1$$

V dalším kroku dělíme r_1/r_2:

$$r_3 = 8.x^2+14.x \qquad q_3 = 12.x+10 \qquad u_3 = 14.x+9$$

V tuto chvíli již polynom r_3 je stupně $t=2$, v dalším kroku nám stačí spočítat jen:

$$u = u_4 = 4.x^2+12.x+14$$

Hodnota koeficientu c je $14^{-1}=3$. Výsledkem tak je

$$\Lambda(x) = u.c = 12.x^2+7.x+1 \qquad \text{a} \qquad \Omega(x) = r_3.c = 11.x^2+x$$

Dosazováním všech hodnot α^i do $\Lambda(x)$ zjistíme, že kořeny jsou:

$$\Lambda(\alpha^{13}) = 12.13^2+7.13+1 = 0 \qquad \text{a} \qquad \Lambda(\alpha^{11}) = 0$$

Inverzí kořenů získáme polohu chyb $\alpha^{e_1}=\alpha^{-13}=\alpha^2$, $\alpha^{e_2}=\alpha^{-11}=\alpha^4$. Víme tedy, že chyby jsou dvě a nastaly na pozicích 2 a 4. Chybový polynom bude tvaru $e(x)=y_1.x^2+y_2.x^4$ a zbývá nám spočítat hodnoty chyb. To provedeme Forneyho vzorcem pro $b=0$:

$$y_j = \alpha^{e_j}.\frac{\Omega(\alpha^{-e_j})}{\Lambda'(\alpha^{-e_j})}$$

Derivace $\Lambda(x)$ je $\Lambda'(x)=7$. Po rozepsání rovnice získáváme vztah pro výpočet velikosti chyby:

$$y_j = \frac{11.\alpha^{-e_j}+1}{7}$$

Pro $e_1=2$:

$$y_1 = \frac{11.\alpha^{-2=13}+1}{7} = 1$$

Pro $e_2=4$:

$$y_2 = \frac{11.\alpha^{-4=11}+1}{7} = 3$$

Teď již můžeme opravit přijatou zprávu:

$$v(x) = r(x)-e(x) = r(x)+1.x^2+3.x^4 = 14.x^7+5.x^6+2.x^5+14.x^4+8.x^3+8.x^2+4.x+14$$

Kapitola 3

Technologické pozadí

Tato kapitola stručně popisuje cílovou platformu této práce a rozhraní Ethernet. Definuje zabezpečovací – RS-FEC vrstvu ethernetového standardu 100GBASE-SR4 výběrem relevantních informací z ethernetového standardu. Cílová platforma je FPGA čip Xilinx Virtex-7[1], konkrétněji XC7VH580T, jenž je srdcem hardwarově akcelerované rekonfigurovatelné síťové karty COMBO[2] projektu Liberouter.

3.1 Technologie FPGA

Programovatelná hradlová pole (FPGA z angl. Field Programmable Gate Array) jsou technologií integrovaných číslicových obvodů, které je možno opakovaně uživatelsky konfigurovat. Tedy jejich funkce není pevně dána při výrobě. Toho se využívá pro opravu chyb, nebo přidávání nových funkcionalit do zařízení obsahující FPGA. Zásadně se tak FPGA odlišuje od obvodů typu ASIC, jejichž logická funkce je pevně dána v okamžiku výroby. A oproti obecným procesorům poskytují velkou míru paralelismu, dokáží tedy na rozdíl od nich zpracovávat velké množství dat a dosahovat tak potřebné propustnosti. V současnosti jsou dva významní výrobci FPGA čipů a to Xilinx a Intel (bývalá Altera). Potom několik menších jako např. Lattice věnujících se hlavně FPGA pro specifické využití.

Nejmenším konfigurovatelným blokem FPGA je *CLB* (Configurable Logic Block) [8]. FPGA tvoří matice těchto bloků, které u Virtexu 7 obsahují 2 funkčně nezávislé podbloky zvané *slice*. Vstupy a výstupy CLB jsou mezi sebou propojovány, a tak tvoří funkčně složitější celky. CLB může být s některým sousedním CLB propojen napřímo, nebo s jiným pomocí propojovací matice (omezené kapacity). Každý slice obsahuje 4 *LUT* (vyhledávací tabulky, také funkční generátory), které provádí libovolnou 6vstupou Booleovu funkci, nebo libovolnou méněvstupou Booleovu funkci s 2 výstupy. Dále obsahuje 8 registrů, multiplexory a pomocnou logiku pro aritmetiku. Můžou tedy mít funkci aritmetickou, logickou, nebo jako ROM paměť. Některé slicy obsahují kromě LUT ještě distribuovanou RAM nebo posuvné registry. Samotné FPGA má pak ve výbavě mnohé další komponenty, často podle specifického cílení konkrétního čipu na určitou doménu.

Z hlediska této práce je podstatné zmínit existenci komponenty transceiveru použitelnou jako Ethernetové rozhraní FPGA čipu. Tyto komponenty nazvané Gigabit Transceivers Xilinx umisťuje na svoje FPGA čipy ve variantách GTP, GTX, GTH a GTZ [9]. Nejrychlejší GTZ a GTH dosahují rychlostí 28,05 Gb/s a 13,1 Gb/s, přičemž cílové XC7VH580T má až 8× GTZ a 48× GTH transceiverů. Dvě čtveřice GTZ transceiverů společně dosahují rychlosti 100 Gb/s v obou směrech

[1] https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html
[2] https://www.liberouter.org/technologies/cards/

uvažovaného Ethernetu. K dosažení nižší rychlosti se používá metoda *převzorkování* (angl. oversampling). Vysílač v transceiveru funguje fundamentálně jako serializér s nastavitelným převodním poměrem 16 až 160 po různě odstupňovaných krocích. To znamená, že v jednom taktu přijme např. 160 bitů dat a do dalšího taktu je stihne sériově odeslat na výstup. Přijímač pracuje obráceně.

Logická funkce nejen rekonfigurovatelných hardwarových obvodů se popisuje pomocí *HDL jazyků* (z angl. Hardware Description Language), mezi které patří i jazyk *VHDL* v němž bude práce implementována. Dalším takovým je jazyk Verilog. Konfiguraci FPGA se často říká firmware a vzniká *syntézou* HDL kódu, při které se převede popsaná funkce na kaskádu funkčních bloků FPGA. Ty jsou poté rozmístěny do konkrétních CLB a korektně propojeny. Tato část syntézy se nazývá *rozmístění a routování* (angl. place & route).

3.2 Ethernet

Ethernet je rodina standardů síťových rozhraní a protokolů vyvíjených v rámci pracovní skupiny 802.3[3] mezinárodní standardizační organizace IEEE. Éter obsažený ve jméně představuje datový kanál, ke kterému se síťové zařízení připojují, aby spolu mohli komunikovat. První 2 verze byly standardizované v roce 1980 a umožňovaly komunikaci rychlostí 10 Mb/s. Mají systematický název 10BASE2 a 10BASE5, kde číslo 10 značí onu přenosovou rychlost a číslo 2/5 maximální komunikační vzdálenost ve stovkách metrů. V obou případech se jednalo o sdílené přenosové médium (point-to-multipoint) a byl jím koaxiální kabel, pouze větší tloušťky u varianty s 500m dosahem. Tato podkapitola čerpá především ze samotného Ethernetového standardu [10].

[11] Ethernet se projevil jako levná a výhodná technologie pro připojování koncových zařízení do lokálních sítí (LAN). Postupně přibývaly nové standardy, v nichž šlo zejména o navyšování přenosové rychlosti. Protože původní varianty měly sdílené médium, byla sdílená i přenosová rychlost, a to všemi stanicemi připojenými k ethernetovému kanálu. Navíc nebylo možné kanál využít 100%, neboť v takovém kanálu nastávaly kolize, pokud více zařízení začalo vysílat v jednu chvíli. Toto se změnilo ve standardu 10BASE-T z roku 1990, kde se přešlo od point-to-multipoint kanálu k point-to-point. K tomu ještě v roce 1997 přibyl plně duplexní přenos, tak bylo možno dosáhnout deklarované přenosové rychlosti v obou směrech zároveň. Pro 1Gb/s Ethernet už nebyla spolehlivost metalického vedení dostatečná a bylo zavedeno FEC kódování, zároveň byla vytvořena varianta tvořená optickým vláknem, která opravné kódování nevyžadovala.

Dnes již existují standardy s přenosovou rychlosti 100 Gb/s označované jako 100GBASE, opět s dodatkem určujícím kvalitu přenosového média a tím i dosah. Existuje tedy např. 100GBASE-SR4, kde SR (Short Range) značí optický kabel nižší kvality (a tedy s kratším dosahem) a 4 je počet linek, po kterých jsou data paralelně přenášena. Ve skutečnosti jsou tedy data přenášena rychlostí 25 Gb/s po každé lince, tedy symbolová rychlost datového kanálu je 25 GBd, přičemž jeden přenesený symbol odpovídá 4 bitům. U této rychlosti došlo k zavedení RS-FEC opravného kódování, včetně některých optických variant, které se při této rychlosti už přestávají chovat jako spolehlivé médium.

[12] Dnes se nepoužívá Ethernet pouze pro lokální sítě, kde funguje nejčastěji ve variantách 10/100BASE, ale i pro větší sítě a zejména síťoví operátoři používají nejrychlejší varianty. V době psaní této práce se dokončují specifikace nových variant 200GBASE a 400GBASE a měly by být oficiálně vydány jako standard 802.3bs v prosinci 2017. Výzkumná komise, která má za úkol vytvořit 400Gb/s Ethernet, vznikla už na začátku roku 2014. Časem však přibyla také varianta 200 Gb/s. Zřejmě proto, že někteří síťoví operátoři už používají různé proprietární verze 200Gb/s Ethernetu,

[3] http://www.ieee802.org/3/

což ukazuje na poptávku i po této variantě. Co se týče symbolové rychlosti, počítá se s 10–50 GBd a nepotvrzeno je 100 Gbd.

3.3 100Gb/s Ethernet

V době psaní této práce existují následující varianty 100GBASE. Liší se dosahem, počtem linek, přenosovým médiem a použitím RS-FEC (varianty s tímto zabezpečením podtrhnuty)[10]:

- 100GBASE-CR4: médiem je stíněný symetrický měděný kabel, dosah nejméně 5 m
- 100GBASE-CR10: větší dosah a bez použití RS-FEC
- 100GBASE-KP4: použití v backplanech (propojovací systém), čtyřstavová modulace signálu pro každou linku[4]
- 100GBASE-KR4: oproti předchozímu pouze dvoustavová modulace
- 100GBASE-SR10: dosah alespoň 100 m přes MMF[5]
- 100GBASE-SR4: přidává zabezpečení RS-FEC
- 100GBASE-LR4: dosah alespoň 10 km přes SMF[6]
- 100GBASE-ER4[7]: dosah alespoň 40 km přes SMF

Popisovat podrobně celý 100Gb/s Ethernet by bylo nad rámec této práce. Pro ni podstatné části Ethernetového standardu, zejména z pohledu implementované varianty SR4, budou probrány v této podkapitole. RS-FEC vrstva je podrobněji popsána v další podkapitole. Ta je téměř stejná u všech variant, takže výsledek této práce bude použitelný s malými, nebo žádnými úpravami i na ostatní zvýrazněné varianty.

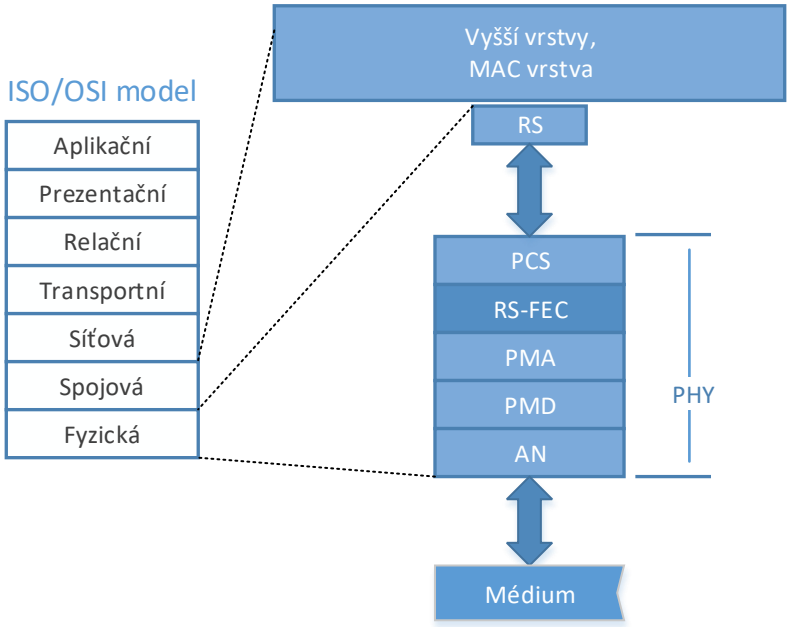
Na obrázku 3.1 je možno si prohlédnout uspořádání vrstev odpovídajícím fyzické vrstvě (zkracováno jako L1) referenčního modelu. Tyto dohromady zajišťují překódování dat od vyšších vrstev, nebo pro ně, tak aby jich bylo možné vyslat do datového kanálu, nebo z kanálu přijat a zpracovat. O vrstvě bezprostředně zapojenou nad určitou vrstvou v architektuře hovoříme, že je jejím *klientem* (např. PMA je na obrázku klientem PMD). Kromě RS-FEC vrstvy, jejíž funkce je již jasná, následuje stručný popis činnosti ostatních vrstev.

Reconciliation Sublayer (RS) zajišťuje převod mezi sériovým datovým proudem formátu MAC a paralelním formátem PCS vrstvy.

Vrstvy seskupené do bloku PHY tvoří zařízení fyzické vrstvy (PHYsical layer device) a to je závislé od konkrétní varianty Ethernetu. Kdežto RS vrstva je stejná pro všechny varianty Ethernetu dané rychlosti.

Physical Coding Sublayer (PCS) je nejsložitější mandatorní vrstva PHY. Vytváří 64b bloky z oktetů přijatých z RS, ty skrambluje pro důvody uvedené v kapitole 2. Pro každý blok provádí překódování 64b/66b, aby na začátku bloku byla 2b synchronizační hlavička (obsahuje změnu logické úrovně – *hranu*). Hrana se tak ve vysílaném proudu na určité pozici zaručeně a periodicky opakuje, což příjemce využije na detekci začátku bloku v příchozím proudu, tedy na synchronizaci. Bloky cyklicky distribuuje (round-robin) do 20 paralelních spojů vedoucích do FEC, nebo PMA vrstvy, dle varianty Ethernetu. Do datového proudu navíc pravidelně, každých 16 383 bloků, vkládá tzv. *zarovnávací značky* (AM), což je blok identifikující každý ze 20 spojů. V přijímací cestě tyto

[4] Při vyslání 1 symbolu dojde k přenosu 2 bitů na každé z linek.
[5] Mnohovidové optické vlákno (z angl. Multi-Mode Fiber) používané na kratší vzdálenosti.
[6] Jednovidové optické vlákno (z angl. Single-Mode Fiber) používané pro delší vzdálenosti.
[7] Short, Long, Extended Range



Obrázek 3.1: Architektura 100GBASE Ethernetu v kontextu vrstev referenčního modelu ISO/OIS

značky vyhledává a po nalezení přeskládává bloky přijaté z jednotlivých spojů do správného pořadí pro RS vrstvu. Kvůli operacím multiplexování nad bity, které se v PMA provádí, je to nutné. Není totiž zaručeno, že blok přijde na tentýž spoj, na který byl původně vyslán. Kromě toho, sobě odpovídající si bloky se můžou na jednotlivých spojích nacházet u příjemce s různým časovým posunem, PCS tyto prodlevy pro RS vrstvu vyrovnává (angl. provádí deskew). Do AM dále vkládá kontrolní součet všech bloků vyslaných před značkou, vypočtený určeným CRC kódem. Na přijímací straně pak informuje vyšší vrstvy o případných chybách.

RS-FEC vrstva není přítomna u všech variant. Vyžaduje 20 vstupních spojů a 4 výstupní, odpovídající 4 fyzickým linkám.

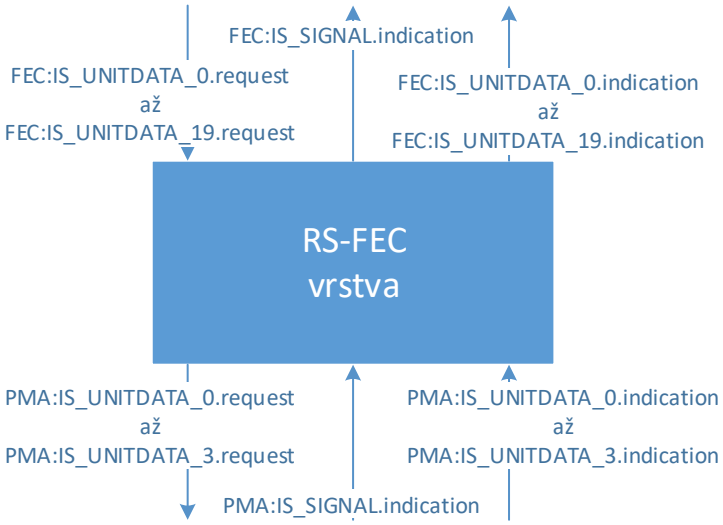
Funkce PCS je obecná pro 100GBASE Ethernet. Nezávislost na médii, kterým se jednotlivé varianty liší, ji zajišťuje vrstva *Physical Medium Attachment* (PMA). PCS je díky tomu kompatibilní s celou řadou fyzických médií. Funkcí PMA je přizpůsobení datového proudu z n abstraktních, nebo skutečných linek na m linek bitově orientovaným multiplexováním, přičemž n a m musí být soudělná. V 100GBASE se jedná o použití 20:10 nebo 20:4, tj. přizpůsobení 20 abstraktních PCS linek na 10, resp. 4 fyzické. PMA však je generická komponenta a je ji možno použít i mezi jinými vrstvami v rámci PHY než jsou PCS/RS-FEC a PMD. Typicky v situaci, kdy funkce jednotlivých vrstev nevykonává jedno zařízení.

Physical Medium Dependent (PMD) je vrstva odpovědná za buzení a snímání signálu na komunikačním médiu. Tato vrstva se od sebe u jednotlivých variant zásadně liší. PMD každé varianty má specifikovány požadavky na elektrické, nebo optické vlastnosti (příp. jiné fyzikální vlastnosti), které musí být splněny. Prakticky tato vrstva odpovídá transceiveru.

Auto-Negotiation (AN) je vrstva zajišťující mezi dvěma komunikujícími ethernetovými zařízeními vzájemnou výměnu informací o schopnostech přenosu dat a na základě toho zajišťuje jejich automatickou konfiguraci. U 100Gb/s Ethernetu je využívána všemi variantami s metalickým vedením. Např. pro domluvu, zda komunikace bude probíhat s FEC zabezpečením, nebo bez něj.

3.4 100GBASE RS-FEC

Rozhraní RS-FEC vrstvy, nacházející se mezi PCS a PMA vrstvou je na obrázku 3.2. PCS je buď přímo jejím klientem, nebo zprostředkovaně přes 1 či více PMA. Obrázek znázorňuje pouze ty signály, které jsou normou vyžadovány jako povinné. Podkapitola popisuje pouze povinné vlastnosti standardu. Přídomek *request* značí signál putující ve směru vysílání, naopak *indication* ve směru příjmu. `FEC:IS_UNITDATA_x` představuje 20 datových linek vedoucích z (do) nadřazené PCS při symbolové rychlosti 5,15625 GBd a `PMA:IS_UNITDATA_x` 4 linky vedoucích do (z) podřazené PMA při 25,78125 GBd. Vysílaný signál `IS_SIGNAL.indication` informuje o tom, zda je obsah `IS_UNITDATA_x.indication` v dané chvíli validní.



Obrázek 3.2: Rozhraní vrstvy RS-FEC bez nepovinných signálů

Následuje popis jednotlivých funkcí vrstvy v pořadí, v jakém se vykonávají na vysílací straně.

Bloková synchronizace linek: Na vysílací cestě dojde nejprve k blokové synchronizaci pro každou z PCS linek – dosáhne se *block locku*, kdy každých 66 za sebou přijatých bitů utvoří 66b blok. Funkce je stejná jako u blokové synchronizace přijímací strany v PCS.

Zarovnání bloků: Poté je každý blok prohledáván na zarovnávací značky, po jejichž nalezení dochází k identifikace každé z linek (*alignment lock*). Po identifikaci všech linek dojde k vyrovnání časového posunu mezi nimi. Přičemž musí být zvládnuto vyrovnat až 49ns maximální posuv mezi linkami. Následně jsou bloky seřazeny ve správném pořadí dle zarovnávacích značek. Samotné bloky s AM jsou přeskočeny a podstupují se jinému zpracování popsanému dále.

Transkódování: Čtveřice bloků z každé po sobě jdoucí čtveřice seřazených PCS linek se spojí v jeden 264b blok. Každý 66b blok buď přenáší data (DATA) nebo příkaz (CMD), což se rozezná typem hrany v hlavičce, případně je hlavička poškozená. V dalším kroku se provádí *transkódování 64/66B na 256/257B*. Tímto dochází k uvolnění místa v datovém toku pro kontrolní symboly zprávy. Označíme-li jako X_i informační část i. bloku (tj. bez prvních 2 bitů hlavičky) a X je D pro DATA bloky a C pro CMD bloky, dále jako X' označíme X s vynecháním 2. niblu (tj. bez bitů 4–7).

Potom transkódovaný blok `tx_xcoded` můžeme popsat jako:

$1D_1D_2D_3D_4$ pokud jsou všechno DATA bloky, nebo:

$01111X_1'X_2X_3X_4$ jeden z bloků má nevalidní hlavičku, jinak:

$0j_1j_1j_2j_3Y_1Y_2Y_3Y_4$ kde $j_i = 0$ pokud i. blok je CMD a $Y_i = C_i'$ pro první CMD, jinak $Y_i = X_i$

Výsledný blok je nazvaný `tx_xcoded`. Transkódování je dokončeno skramblováním prvních 5 bitů, čímž vzniká 257b blok `tx_scrambled`.

Mapování (zarovnávacích) značek: Vyjmuté zarovnávací značky jsou přemapovány tak, aby je protistrana byla schopna detekovat i na chybovém kanálu, provést zarovnání a seřazení. Značkám prvních 4 i posledních 4 linek jsou pozměněny oktety identifikátoru, aby se identifikovaly jako linka 0, resp. 16. Tak stačí příjemci vyhledávat na každé ze 4 linek značku odpovídající lince 0 (na místo značek 0–3) na získání blokové synchronizace. Z některého z dalších 3 bloků je potom schopen zjistit, které FEC lince odesílatele daná linka odpovídá. Značky jsou poté mapovány takovým způsobem, aby i po průchodu symbolovou distribucí byly vyslány na správnou FEC linku v původním formátu PCS. Na FEC-0 (FEC linka 0) značky z linek $0, 4, 8, 12, 16$, na FEC-1 z $1, 5, 9, 13, 17$, atdl. Tyto bloky jsou namultiplexovány po 10 bitech do `am_txmapped`, čímž se kompenzuje pozdější symbolová distribuce (viz dále). Do bloku je přidána ještě pětice bitů a zakódována tak, aby značkami začínal zabezpečený blok RS kódu. Pětice bitů se invertuje při každém vložení značek. Slouží tak pro získání blokové synchronizace FEC linek příjemce.

Reed-Solomonův enkodér: RS-FEC vrstva zabezpečuje zprávu o délce 5140 bitů, kterou tvoří 4 po sobě jdoucí pětice bloků `tx_scrambled`. V SR4 variantě (a také CR4 a KR4) se využívá kód $RS(528, 514)$ nad $GF(2^{10} = 1024)$ s generujícím a primitivním polynomem $G(x) = x^{10} + x^3 + 1$ a $\alpha = x^3 + 1$. Zabezpečený blok má tedy délku 5280 bitů, neboť kód pracuje nad 10bitovými symboly. Vytvářecím mnohočlenem je:

$$\begin{aligned} g(x) &= \prod_{i=0}^{r-1}(x - \alpha^i) = \\ &= x^{14} + 904x^{13} + 6x^{12} + 701x^{11} + 32x^{10} + 656x^9 + 925x^8 + \\ &\quad 900x^7 + 614x^6 + 391x^5 + 592x^4 + 265x^3 + 945x^2 + 290x + 432 \end{aligned} \tag{3.1}$$

Schéma enkodéru je stejné jako na obrázku 2.4.

Kód je schopný detekovat a opravit $t = 7$ symbolů. Dekodér by ale dle normy měl umět pracovat i v režimu, kdy chyby pouze detekuje pro snížení latence v souvislosti s opravou chyb. V případě, že pracuje v tomto režimu, nebo detekuje neopravitelnou chybu, záměrně poškodí synchronizační hlavičky 66b bloků odesílaných do PCS vrstvy, tak aby způsobil, že PCS zahodí všechny bloky, které se alespoň částečně nacházeli v přijatém neopravitelném bloku jako neplatné.

Symbolová distribuce: Zabezpečená zpráva je odeslána na výstup do PMA symbolovou (tj. po 10 bitech) cyklickou distribucí do 4 fyzických FEC linek.

Strana příjemce pouze provádí opačnou činnost než strana odesílatele. Podrobnější popis jednotlivých funkcí RS-FEC vrstvy je v ([10], klauzule 91).

Kapitola 4

RS-FEC vrstva vysílací strany

Vzhledem k tomu, že vrstva RS-FEC funguje nezávisle na vysílací (Tx) i přijímací (Rx) cestě, je přirozeně i samotný návrh rozdělen do dvou kapitol. Tato kapitola se zabývá obecným návrhem Tx strany, který je vypracován v souladu s normou Ethernetu, jíž se zabývá předchozí kapitola. Dále provedenými úpravami v návrhu pro implementaci a integraci do LR4 varianty Ethernetu v rámci projektu Liberouter. Jsou zmíněny i výsledky této implementace a návrhy na další možná vylepšení.

Pro výhodnost jsou zavedeny některé další pojmy a konvence. *Oktet* a *nibl* označují osmici, resp. čtveřici po sobě jdoucích bitů. V zápise signálů s šířkou větší než 1 bit jsou bity indexovány zleva, příp. nejlevější odpovídá prvnímu přenášenému bitu. Názvy signálů jsou pokud možno stejné, jako jsou v normě, případně v implementaci. Pod pojmem *design* se myslí existující implementace.

4.1 Obecný návrh

Klíčovými pro návrh nejsou jenom funkce vrstvy popsané v předchozí kapitole, ale i parametry, které musí návrh splňovat. To jsou:

- Schopnost vyrovnat až 49ns relativní posun (skew) přijatých bloků mezi linkami PCS
- Informační propustnost 100 Gb/s, vzhledem k překódování 64b/66b však 103,125 Gb/s

Pochopitelným dalším požadavkem pro reálný provoz je co nejnižší zpoždění dat na Tx cestě, které přidání RS-FEC vrstvy způsobí (tzv. low-latency design).

Naopak nejsou kladeny normou přesné požadavky na klientské rozhraní vrstvy, pouze požadavek napojení pro 20 PCS linek. To znamená, že je volnost samotného návrhu rozhraní, resp. šířky těchto linek a ruku v ruce i pracovní frekvenci obvodu. Jako první krok návrhu je tedy volba vhodného rozhraní, resp. šířky. Vodítkem pro její volbu může být činnost samotné vrstvy. Před zakódováním RS kódem se provádí serializace datových proudů z PCS linek, a to po 66b blocích. Není proto užitečné pracovat s datovou šířkou linky menší než je tato velikost bloku (resp. na všech linkách 20×66 b). Kodér totiž musí vyčkat příchodu celého 1. bloku (z PCS linky 0 – PCS-0), než začne zpracovávat další, 2. blok (z PCS-1). Vyčkat celý 2. než začne zpracovávat 3., atd. Jenže tyto bloky (všech 20) jsou k dispozici všechny současně, a tak je i všechny současně může v tu chvíli zpracovat. Práce s nižší šířkou by znamenala nutnost použití nadbytečného množství paměťových prvků sloužících jako vyrovnávací paměť pro odesílání. Přičemž by se tím nedosáhlo nižšího zpoždění průchodu Tx RS-FEC, jak by se dalo intuitivně očekávat. Jinými slovy, na FEC linky je v jeden moment vysílán proud jediné PCS linky, ale 20× vyšší rychlostí. V časových jednotkách odpovídá přijetí 1 bloku (66 bitů) přesně 12,8 ns a je to hodnota teoretického minima zpoždění způsobeného RS-FEC vrstvou na Tx cestě.

Návrh je proto dělán pro tuto šířku – odpovídá ji frekvence $f = \frac{100\text{ Gb/s}}{20\times 64\ b} = \frac{103{,}125\text{ Gb/s}}{20\times 66\ b} \doteq 78$ MHz. Avšak dává smysl zabývat se i šířkami linek, jež jsou 2násobkem, nebo 4násobkem zvolené šířky – nepřímo úměrně tomu se sníží i požadovaná frekvence obvodu, ale zvýší se tím jeho zpoždění.

Schéma návrhu je na obrázku 4.1. Bílý zobáček na vstupu značí přívod náběžné hrany hodinového signálu, jelikož tyto komponenty jsou stavové a řízeny konečnými automaty. Jinak jsou však tyto bloky autonomní ve smyslu, že si kromě signálu AMI svůj stav řídí samy pouze na základě jimi procházejících dat a nevyžadují ani reset.

Synchronizace: Řeší zarovnání bloků na jednotlivých linkách, tak aby na svém výstupu měla každý takt připravena jediný a celý 66b blok pro každou linku, počínaje synchronizační hlavičkou (první dva bity 01 – DATA, 10 – CMD). Jednotka pracuje pro každou linku zvlášť a postupuje tak, že pro každou pozici 0–65 v bitovém proudu zkouší, zda se na ní hlavička nachází. Pokud se povede na dané pozici nalézt validní hlavičku 64 bloků po sobě, dosáhla linka block locku (BL). Pokud ne, zkouší to na vedlejší pozici (provede se *„slip“*). Aby zahájila následující jednotka hledání značky pro konkrétní linku až po jejím blokovém zarovnání, záměrně pro ni do té doby zneplatňuje výstupní hlavičky. I po získání BL dále pokračuje v kontrolování a nalezne-li 64 a víc neplatných hlaviček v sekvenci 1024 bloků, zarovnání bylo ztraceno a hledání se pro danou linku restartuje.

Řešení operace slip záleží na konkrétním způsobu připojení klientské vrstvy. Pokud je to přes receiver s deserializérem, může se provést změnou jeho fáze. V ostatních případech se použije kupř. bitových posuvů.

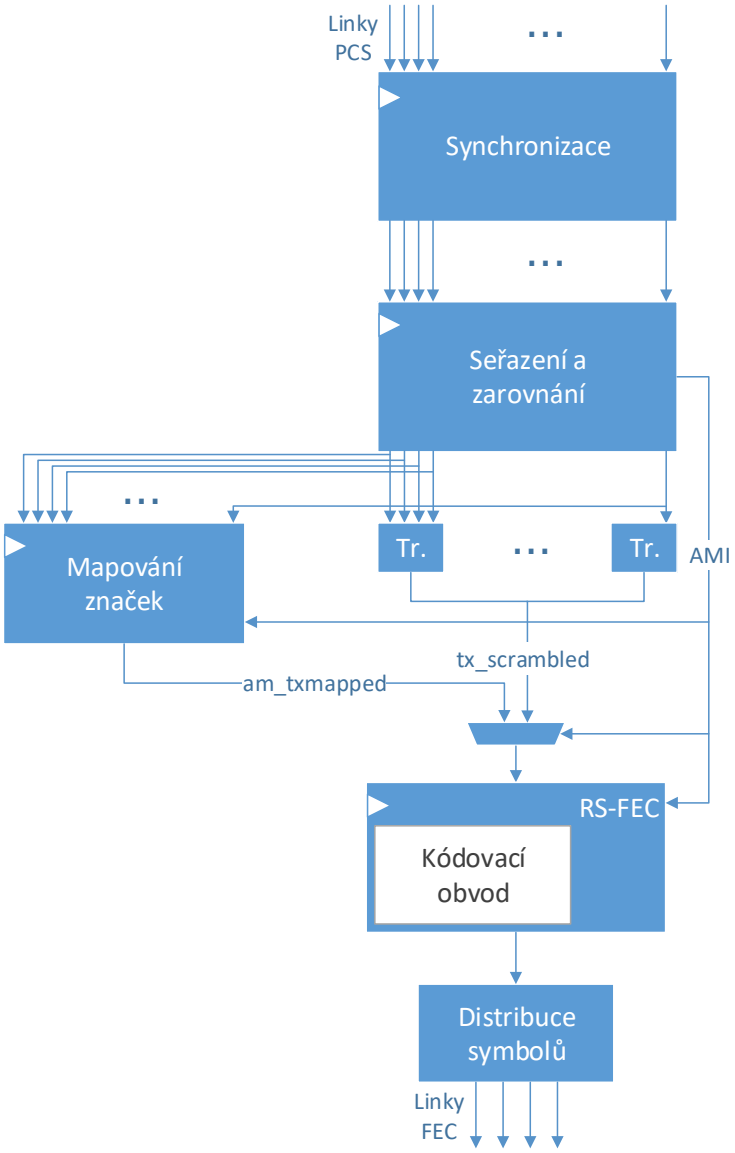
Příklad možného postupu hledání počátku bloku v bitovém datovém proudu je na obrázku 4.2. Nejprve je hledána hlavička na výchozí pozici označené jako 0, kde nalezena není. Proto je proveden slip a nový pokus je proveden o 66+1 bitů dále – na pozici 1 vůči výchozí pozici. Tam nalezena je, slip se neprovádí a další test je o 66+0 bitů dále – stále pozice 1. Takto se to opakuje až po pozici 3, na které je hlavička nalezena 64× po sobě a je dosáhnuto BL. Správná pozice je tedy o 3 bity dále vůči výchozí pozici.

Pro obvody, kde je PCS i FEC na jednom čipu (např. FPGA), nebo jakékoliv jiné obvody, kde se dá zaručit, že se nevyskytnou v datových proudech jednotlivých linek posuvy, není tato jednotka potřebná.

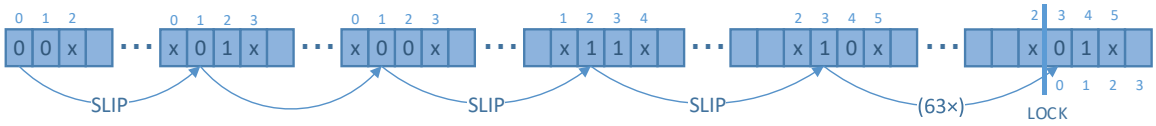
Seřazení a zarovnání bloků: Tato jednotka má 3 úkoly. Zajišťuje zarovnání datových proudů na všech linkách tak, aby si výstupní bloky odpovídaly (pokud nepřekročí jejich relativní posun víc než normou stanovených 49 ns). Dále je seřazuje podle zjištěného čísla linky. A v zarovnaném stavu informuje ostatní komponenty o tom, zda má zrovna na výstupu bloky AM signálem AMI (AM Indicator). Jednotka bude podrobněji probrána v následující podkapitole.

Transkodéry: Zajišťují překódování 64/66B na 256/257B pro čtveřice bloků z po sobě jdoucích linek. Jedná se o soustavu multiplexorů řízených hlavičkami těchto vstupních bloků. Provádí také skramblování prvních 5 bitů operací xor s bity 8–12.

Mapování značek: Ačkoliv norma popisuje vyjímaní značek z datového proudu, jejich mapování a znovuvložení jako 3 kroky, je možné to udělat i v rámci jednoho kroku a tato jednotka je tak navržena. Jednotka pracuje paralelně k transkodérům. Její užitečná činnost spočívá v mapování AM v momentě, kdy dorazí z PCS. Jenom v tu chvíli je její výstup přiváděn přes multiplexor jako část zprávy do enkodéru. Veškerý ostatní čas je svými výstupy odpojena. V podstatě tak vyjmutí značek představuje jejich přivedení na vstup této jednotce, kde jsou namapovány a znovuvložení



Obrázek 4.1: Schéma vysílací strany RS-FEC vrstvy



Obrázek 4.2: Ukázka postupu blokové synchronizace

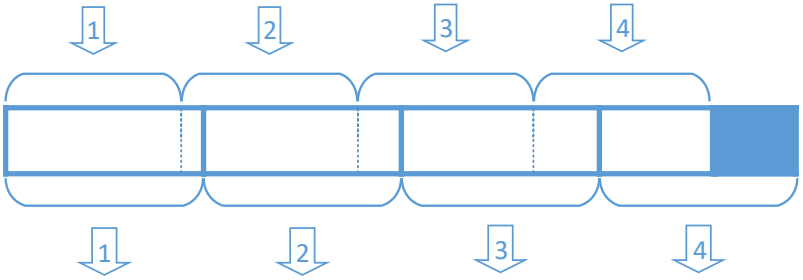
představuje přepnutí vstupu enkodéru. Se samotnými zarovnávacími značkami obsaženými v blocích není potřeba nic dělat na linkách 0 a 4–16, kontrolní součty jim zůstávají a norma popisuje vložení oktetů, které už v nich jsou. Bity kontrolního součtu uložené v blocích jsou ponechány beze změny i zbývajícím linkám (oktety BIP). Přečíslování linek 1–3 na 0 a 17–19 na 16 se děje přepsáním identifikující části (oktety M_0,M_1... a M_6) značky, a to hodnotami z odpovídající části značky pro linku 0, resp. 16. Pozice zmíněných oktetů uvnitř značky jsou znázorněny v tabulce 4.1.

10	M_0	M_1	M_2	BIP_3	M_4	M_5	M_6	BIP_7

Tabulka 4.1: Formát 66b bloku zarovnávací značky (kromě 2bitové hlavičky po oktetech)

Pozměněné značky bez synchronizačních hlaviček jsou poté setříděny do 4 bloků `am_txpayloads_x` tak, že nultý obsahuje za sebou značky z linek $0, 4, 8, 12, 16$, první $1, 5, 9, 13, 17$, atdl. A tyto bloky jsou serializovány po 10 bitech do signálu `am_txmapped`, čímž se kompenzuje pozdější symbolová distribuce (viz dále). Tento signál je doplněn 5bitovou výplní, aby měl šířku 1285 bitů stejně jako výstup transkodérů. Výplň má na začátku hodnotu 01011 a s každým příchozích AM (to jednotka pozná ze vstupu AMI) se invertuje. Hodnota výplně je jediný stav této jednotky.

RS-FEC: Kódové slovo sestává ze 4 pětic transkódovaných bloků. Při zvolené šířce linek trvá sestavení celého kódového slova 4 takty (51,2 ns) + zpoždění kódovacího obvodu. Takže je i odesláno po 4 částech. Informační úsek (zpráva) pro výpočet kontrolního úseku však také přichází po 4 částech. Jednotka proto obsahuje vyrovnávací paměť jako je na obrázku 4.3, kde šipky znázorňují vstup jednotlivých části zprávy a výstup jednotlivých části slova. Část na výstupu je o 1 takt opožděna oproti odpovídající části na vstupu. To je umožněno tím, že předchozí vstup jde na výstup přes paměť a aktuální se propaguje na výstup napřímo. Pokud by celý výstup byl přes paměť, jednotka by vytvářela 1 takt zpoždění navíc. Vybarvená část není součásti paměti a značí vypočtený kontrolní úsek. Kódovací obvod (enkodér) bude podrobněji popsán v následující podkapitole a pracuje nad zprávou uloženou v této vyrovnávací paměti. Informaci o aktuálně zpracovávané části udržuje jednoduchý konečný automat. Protože AM musejí být vždy na začátku kódového slova, reaguje automat na signál AMI restartováním – stav zpracování první části.



Obrázek 4.3: Vyrovnávací paměť jednotky RS-FEC

Symbolová distribuce: Rozděluje část kódového slova vystupujícího z enkodéru cyklicky po symbolech do FEC linek. V implementaci může existovat bez jakýchkoliv kombinačních logických členů – tvořená pouze přenosovými cestami připojující bity výstupu jednotky RS-FEC na jejich správnou pozici ve správné FEC lince.

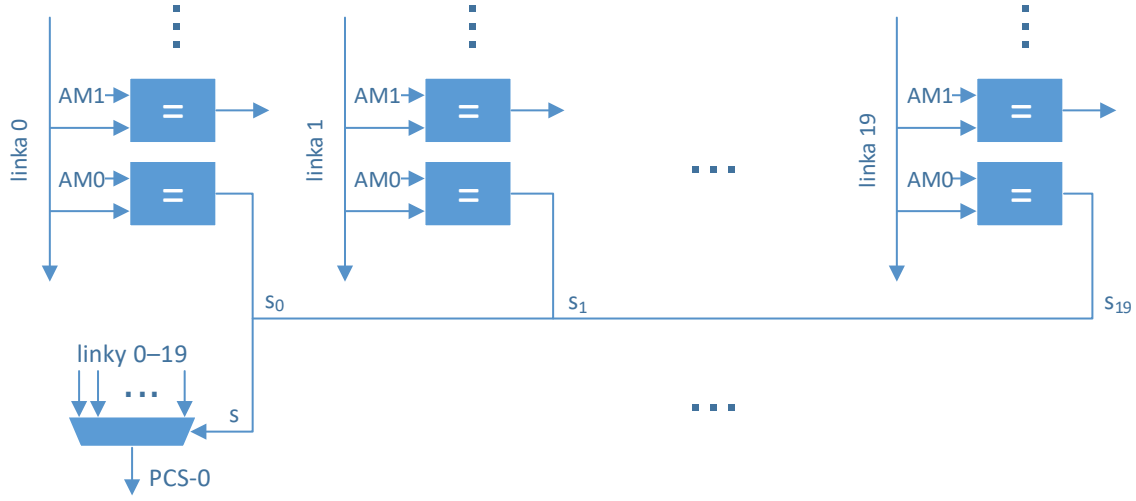
4.2 Návrh jednotky seřazení a zarovnání

Navrhovaná jednotka má na vstupu 20 linek (označeny jako „linka“) bez znalosti, která linka odpovídá které lince výstupu klientského PCS. Z jednotky vystupuje 20 PCS linek (označeny jako PCS), tj. ve správném pořadí. A skládá se ze 3 funkčních bloků pro každou z 20 linek. První má na

starosti detekci AM na lince a její identifikaci podle ní, druhý zajišťuje seřazení linek a třetí jejich vzájemné zarovnání.

Identifikátor linky: Pro každý 66b blok linky se kontroluje, zda odpovídá některému z 20 možných AM. Tedy komparátor porovnává všechny bity kromě BIP oktetů (viz tabulka 4.1) na shodnost. Pokud jeden z 20 komparátorů na jedné lince narazí na shodu, je poznačeno číslo linky a detekce se pozastaví až do příchodu bloku, v kterém je očekávána další, stejná značka. Pokud tam není, identifikace se spouští od začátku. Pokud ano, byly nalezeny dvě stejné značky s rozestupem specifikovaným normou a dochází k alignment locku (AL) – identifikována linka. I po identifikaci probíhá periodická kontrola značky. Pokud 4× po sobě dorazí špatná nebo žádná značka, dochází ke ztrátě locku.

Řadič linky: Každá PCS je vytvářena multiplexováním z 20 vstupních linek. Každý identifikátor má poznačeno číslo linky v kódu one-hot (1 z n) i multiplexor je typu 1 z n a je řízen 20bitovým signálem s. Logická 1 na bitu s_x v tomto signálu odpovídá propuštění linky x přes multiplexor. Signál pro multiplexor PCS linky y je konkatenací bitů na pořadí y z poznačených čísel linky ze všech bloků identifikátorů. Tento systém je naznačen na obrázku 4.4 pro PCS-0. Řadič na něm představuje multiplexor řízený soustavou komparátorů (ty představují detekční část identifikátorů).

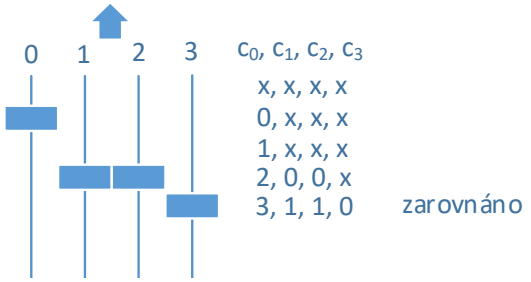


Obrázek 4.4: Seřazení vstupních linek dle zarovnávacích značek

Zarovnání linky: Zarovnání linek je možno provést zpožděním linek, jejichž AM přijdou dříve (v toleranci stanovenou normou) než na ostatních linkách. Seřazené linky jsou každá zapojena do *dynamického posuvného registru*. To je posuvný registr s adresovaným výstupem, neboli dynamickou hloubkou. Jeho (maximální) hloubka odpovídá počtu o jeden většímu, než je PCS bloků, které jsou přijaty za 49 ns, což je maximální možný posun linek vůči sobě. Jeden blok je přijat každých 12,8 ns, takže maximální posun vyjádřený v blocích je 4 a registr má hloubku 5.

Každý z registrů je napojen na svůj čítač 0 až 4. Pokud už na všech linkách platí stav AL, čítání se zahájí detekcí AM na příslušné lince. Touto činností se počítá, o kolik taktů je nutno opozdit výstup linky, aby byl zarovnán vůči nejopožděnější lince. Čítání se úspěšně ukončí momentem,

kdy jsou sepnuty všechny čítače, tj. nastala situace, že během 5 taktů dorazil AM na každou linku. Nastává zarovnaný stav a každý čítač obsahuje hodnotu pro adresování svého registru tvořícího požadované zpoždění. Pokud čítač přeteče, čítání se resetuje a čeká se na příchod další značky. Totéž se stane, pokud některá z linek přijde o AL. Příklad zarovnání pro hypotetické 4 linky je na obrázku 4.5 a značí proud dat na 4 linkách. Každý řádek znázorňuje jeden takt. Obdélník je výskyt AM v daném čase na dané lince. Vpravo jsou čítače pro všechny linky, x značí jeho neaktivní stav. Zarovnání trvá 4 takty a linka 0 je po něm na výstupu opožděna o 3 takty vůči lince 3, linka 1 a 2 o jeden takt vůči lince 3.



Obrázek 4.5: Postup zarovnání pro 4 linky

Zarovnání zanáší do vrstvy dynamické zpoždění 1 až 5 taktů. Signál AMI indikuje AM na výstupu jednotky pouze ve stavu zarovnání.

Pokud jsou vstupní linky vedeny přímo z PCS, nebo je jinak zaručeno správné pořadí, není potřeba řadič ani kompletní identifikátor – postačí jeden komparátor na hledání příslušné značky v čase pro detekci AL na linku. Pokud je kromě toho zaručeno i to, že nebude docházet k relativní posunům bloků vůči sobě, není potřeba ani funkce zarovnání linky a postačuje pouze jediný identifikátor na některé z linek pro generování signálu AMI dalším jednotkám.

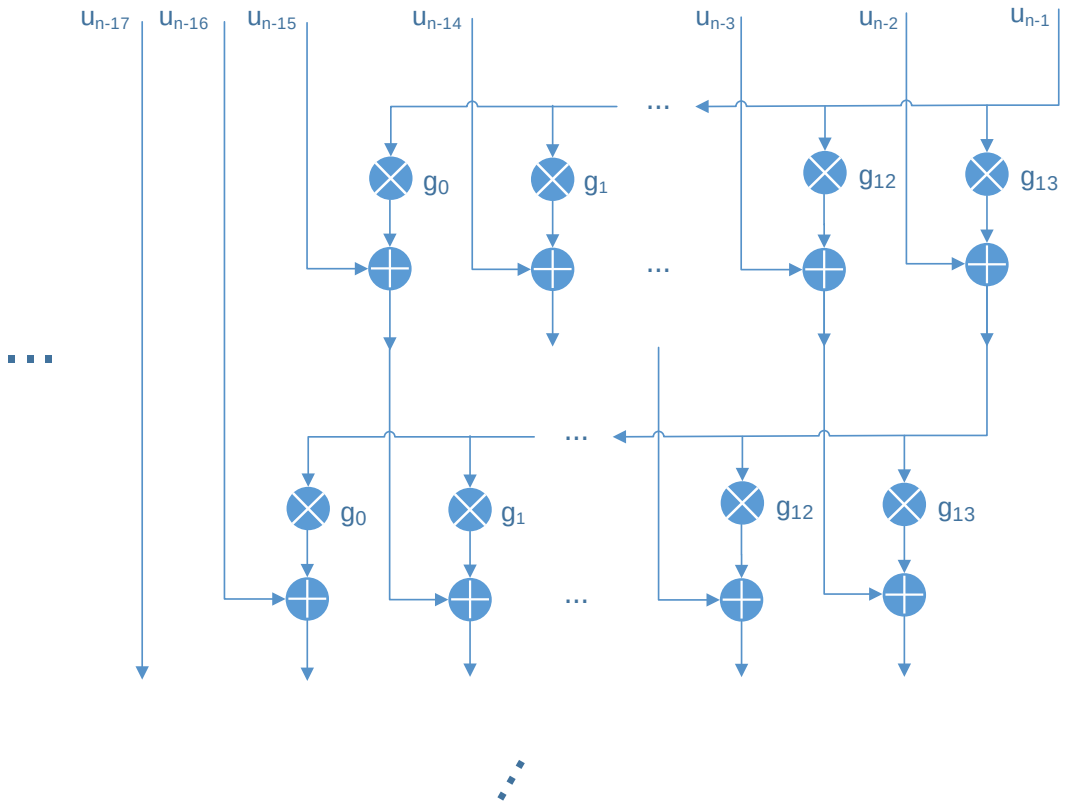
4.3 Návrh kódovacího obvodu

Kódovací obvod zajišťuje výpočet kontrolních symbolů jednotky RS-FEC. Pokud by byl použit kodér podobný sériovému CRC kodéru na obrázku 2.4, zpracoval by během jednoho taktu jediný vstupní symbol. Aby dosahoval požadované propustnosti, jeho pracovní frekvence by musela být $f = \frac{103{,}125 \text{ Gb/s}}{10 \text{ b}} \doteq 10$ GHz. Takový obvod je efektivní z hlediska spotřebované logiky díky tomu, že využívá maximálním způsobem zřetězení. Nicméně dalece převyšuje zvolenou ideální frekvenci v kontextu celé vrstvy podle dříve uvedené úvahy. Navíc převyšuje i reálné možnosti běžné techniky. V případě cílového FPGA Virtex-7 více než desetinásobně[1].

Je tedy nutné vytvořit návrh obvodu, který výpočet provede v méně taktech. Podle inspirace dlouhým dělením lze rozvinout iterační výpočet sériových kodérů do kaskády tak, aby vykonal výpočet zabezpečení nad všemi symboly v jednom taktu. Úrovně kaskády představují sériové kodéry zapojené za sebou s tím rozdílem, že paměťové členy jsou odstraněny a výsledky součtových operací přivedeny přímo do další úrovně kaskády, jak to znázorňuje obrázek 4.6. Principiálně každá úroveň kaskády představuje jeden krok dlouhého dělení. Byla tak odstraněna sekvenčnost původního kodéru a jedná se o kýžený kombinační obvod.

Celkově je kaskáda tohoto obvodu tvořena n úrovněmi. Pro $RS(528, 514)$ je to 514 úrovní. Do kaskády vstupuje 514 symbolů zprávy a na jejím výsledku je 14 kontrolních symbolů. Průchod

[1] Maximální frekvence globálního hodinového signálu $f_{MAX\_BUFG} = 741$ MHz [13].



Obrázek 4.6: Detail paralelního Reed-Solomonova kódovacího obvodu

úrovní pro symbol znamená průchod jednou násobičkou a jednou sčítačkou[2], nebo pouze sčítačkou. První vysílaný symbol, u_{n-1}, prochází od vstupu do obvodu až do výstupu všemi úrovněmi, tj. 514 násobičkami i sčítačkami. Tvoří tak z hlediska počtu logických prvků nejhorší cestu. Součet zpoždění přepnutí logických hradel a transportního zpoždění signálu na této cestě tak musí být nižší, než odpovídá periodě pracovní frekvence. Každý další symbol má při průchodu o 1 násobičku méně než předchozí.

Jak bylo řečeno v kapitole 2.2.1, operace sčítání $\oplus$ je pouhý exkluzivní součet (xor) nad sobě odpovídajícími si bity dvou symbolů. Nemá tak velký vliv na složitost tohoto obvodu oproti složitému součinu $\otimes$. Ten může být implementován rozvinutím násobení i následného dělení modulo, nebo logaritmicky.

4.4 Prvotní implementace

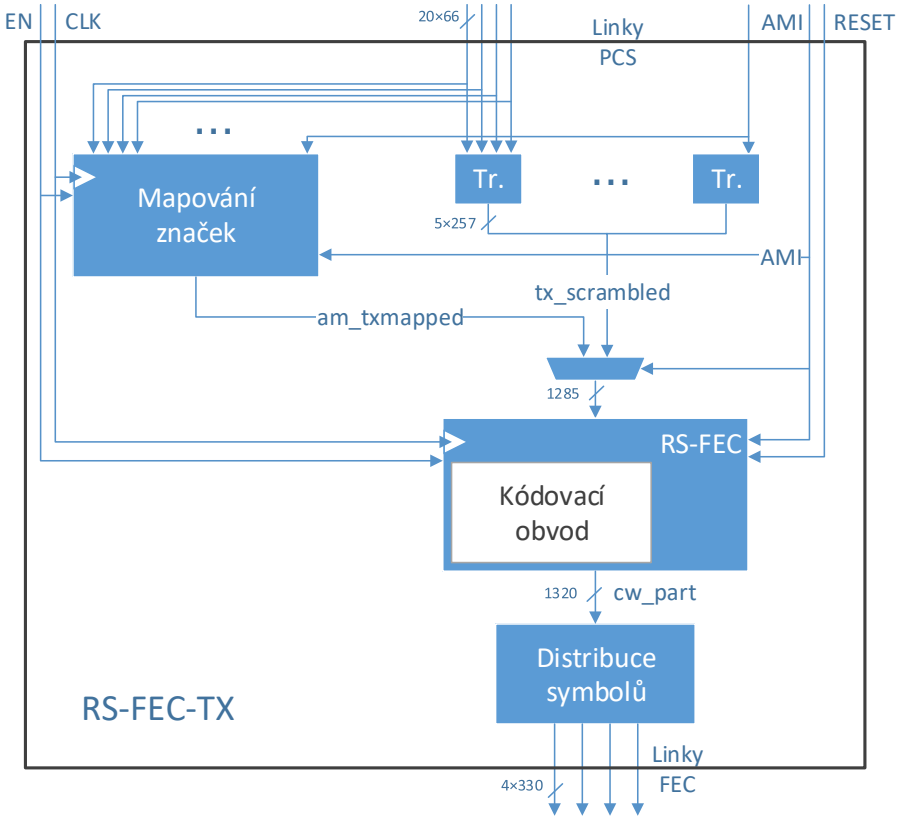
Protože cílovou platformou je FPGA, byl pro implementaci vytvořen zjednodušený návrh. V FPGA je vrstva RS-FEC vysílací strany napojena na PCS vrstvu přímo, bloky na vstupních linkách jsou tak vždy kompletně zarovnané. Jednotka synchronizace proto není implementována a není implementována ani jednotka zarovnání a seřazení linek, a to ani částečně. Díky tomu zůstal nárůst zpoždění průchodu dat Tx stranou vrstvy na jejím teoretickém minimu. Potřebný signál AMI, indikující značky na vstupu jednotky, byl přiveden do jednotky zvenčí. V rámci projektu Liberouter bylo totiž

[2]Zde je míněn součet a součin jako operace v konečném tělese. Násobička a sčítačka jsou funkční bloky vykonávají právě tyto operace.

možné přistoupit ke zdrojovým kódům ostatních vrstev designu Ethernetového rozhraní. Toho bylo využito a vrstva PCS byla upravena vyvedením jejího vnitřního signálu AMI a napojením do vrstvy RS-FEC. Je to v tomto případě nejefektivnější implementace detekce AM. Ačkoliv rozhraní PCS linek v designu Liberouter je stejné jako rozhraní obecného návrhu, pracovní frekvence designu je vyšší než navrhovaná. Na sekvenční jednotky je tak kromě hodinového signálu přiváděn i povolovací signál, jenž je periodicky nastaven v logické 0 a vyrovnává tak rychlosti PHY a ostatních části designu. Z praktických důvodů byl přidán i resetovací signál. Shrnutí podstatných vlastností a změn oproti obecnému návrhu:

- Vstup PCS signálu o šířce 20×66 bitů, odpovídající 20 PCS linkám o šířce 66 bitů
- Výstup o stejné šířce, odpovídající 4 FEC linkám o šířce 330 bitů
- Odstraněny jednotky synchronizace a zarovnání
- Zvenčí přivedeny signály resetu (RESET), povolení (EN) a AMI
- Zpoždění vrstvy ∼12,4 ns (pracovní frekvence 80,5 MHz)

Schéma takto upraveného návrhu je na obrázku 4.7. Zobrazuje i vstupy, výstupy, implementované jednotky a šířky signálů. Jednotlivé komponenty provádějí činnost popsanou v obecném návrhu.



Obrázek 4.7: Implementační návrh Tx strany RS-FEC vrstvy

Implementaci jsem dekompozicí rozdělil na dvě hlavní části a provedl ve VHDL. Pro přehlednost je rozdělena ve 2 samostatných souborech:

- Rozhraní RS-FEC vrstvy a její vnitřní logika v souboru `rs-fec-tx.vhd`. Využívá kódo-

vacího obvodu.

- Kódovací obvod v souboru `rs-fec-enc.vhd`. Implementován dle schématu 4.6 a kontrolní symboly počítá během jednoho taktu. Operace + a · tělesa $GF(10)$ jsem ve VHDL implementoval (pro zachování sémantiky) algoritmicky přetížením funkcí operátorů + a *. Operace součinu byla implementována postupem násobení a následného dělení modula tak, jak bylo uvedeno v kapitole 2.2.1.

Pro syntézu byl použit nástroj Vivado 2016 jakožto oficiální syntézní nástroj pro FPGA čipy firmy Xilinx. Cílová platforma syntézy byla nastavena jako FPGA XC7VH580T používané v projektu Liberouter. Syntetizovaný design spotřebuje 52 089 LUT (14,3 % na čipu dostupných), 5146 registrů (< 1 %) a doba syntézy trvá 6 h a 32 min.

4.5 Optimalizování implementace

Prvotní implementace je funkční, splňuje časování i využité zdroje jsou zřejmě adekvátní použitému kódu. Nicméně doba, jakou nástroji Vivado trvá syntéza je nezvykle dlouhá a značně převyšuje dobu překladu kompletních designů COMBO karet v projektu nevyužívajících RS-FEC. To by znamenalo nepohodlí pro vývojáře těchto karet. Řešením by bylo používat v designu již vysyntetizovanou vrstvu RS-FEC a ke kompletnímu designu ji použít jako „černou skříňku“, to však syntéznímu nástroji znemožní provádět některé optimalizace. Ačkoliv spotřebované zdroje jsou adekvátní použitému kódu, jejich množství je značné a dává smysl zabývat se jejich snížením. Vezmeme-li v úvahu, že zakódování je jednodušší než dekódování a oprava chyb, dá se počítat, že dohromady by RS-FEC vrstva zabrala nejméně 30 % LUT v FPGA, což výrazně zmenšuje prostor pro jakoukoliv akcelerovanou aplikaci běžící na kartě.

Vzhledem ke způsobu implementace jsem dopěl k těmto možnostem způsobující pomalou syntézu:

- Implementace operací v $GF(1024)$ pomocí funkcí (nebo způsob jejich popisu ve VHDL) je pro HW syntézní nástroje nepřirozené a nedokážou s nimi dobře pracovat.
- Výsledná síť je jednoduše příliš rozsáhlá, nebo příliš hierarchická na to, aby ji syntézní nástroj dokázal efektivně zpracovat.
- Obvod se při své rozsáhlosti nečlení na menší jednotky, což omezuje efektivní lokální optimalizace logické sítě. Toto je zároveň upozornění nástroje Vivado při průběhu syntézy.

Nejprve jsem vyzkoušel několik úprav zápisu kódovacího obvodu ve VHDL, které se zaměřily na zjednodušení použitých funkcí, nebo jejich odstranění. Funkce násobení byla zjednodušena tak, že se v ní již nepočítal obecný algoritmus násobení prvků konečného tělesa, ale byl rozvinut do logické sítě, neboli byli z ní odstraněny cykly (byť s konstantním počtem kroků). Násobení dvou prvků $C(x) = A(x).B(x)$ lze totiž rozepsat na část násobení jako:

$$\begin{aligned} c'_0 &= a_0.b_0 \\ c'_1 &= a_1.b_0 + a_0.b_1 \\ &\vdots \\ c'_{18} &= a_9.b_9 \end{aligned}$$

a na část dělení modulo jako ($G(x) = x^{10} + x^3 + 1$):

$$\begin{aligned} c_0 &= c'_0 + (c'_{10} + c'_{17}) \\ &\vdots \\ c_9 &= c'_9 + c'_{16} \end{aligned}$$

Zjednodušení funkce směrem k jednoduchým logickým operacím v několika verzích však nemělo žádný pozitivní vliv. Předělání designu na použití VHDL entit pro násobičku pak mělo výrazně negativní vliv, kdy kaskáda obvodu nebyla zploštěna, obsahovala tedy extrémně dlouhou kombinační cestu (tzv. *kritickou cestu*), která by při implementaci do HW svým zpožděním signálu několikanásobně převyšovala pracovní periodu FPGA. Změny zápisu kódu se tedy všechny ukázaly jako neperspektivní a nikdy nepřekonaly původní implementaci. Některé zkoušené varianty jsou přiloženy mezi soubory hotové implementace.

Dalším postupem bylo vzhledem k tomu, že navržený kódovací obvod počítá celé zabezpečení v 1 taktu, vytvoření zřetězeného zpracování na základě první implementace. Pohledem na schéma rozvinutého kodéru (obrázek 4.6) o 514 úrovních je zřejmé, že kódovací obvod lze rozdělit na 2 části po 257 úrovních bez velkých úprav. Kontrolní symboly vypočtené v 1. kroku (2. taktu) jsou pouze na začátku 2. kroku (4. taktu) přičteny k prvním přijatým 14 symbolům. Protože však 1 část kódového slova přijde každý takt, bylo by ideální vytvořit zřetězení po 1 taktu namísto po 2 – to znamená po 128,5 symbolech přijatých každý takt. Z definice cyklických kódů přirozeně plyne, že nulový symbol přidaný na začátek zprávy nezmění vypočtený kontrolní úsek: $0.x^k + u(x).g(x) = u(x).g(x)$. Můžeme tak vytvořit děličku o 129 krocích, přičemž bude střídavě plněna 128 symboly doplněnými o nulový symbol na začátku a 129 symboly. Postačuje pouze v lichém kroku uložit koncový půl symbol do dalšího kroku, kde se konkatenuje s dalšími.

Toto řešení poskytlo očekávaný pokles požadavků na zdroje a doba syntézy se také zkrátila, stále však zůstala nezvykle dlouhá. S vedoucím práce, Lukášem Kekelym, jsme došli k závěru, že bude lepší vygenerovat vlastní logický obvod pro výpočet kontrolních symbolů a pouze ten nechat syntetizovat. Vytvořili jsme tedy utvářecí matici pro Reed-Solomonův kód v podobě cyklického nesystematického kódu podle kapitoly 2.2.3. Tu jsme převedli pomocí řádkových operací do kanonické formy a získali jsme tak zabezpečovací podmatici 129×14 systematického cyklického Reed-Solomonova kódu s koeficienty v $GF(1024)$. Z této matice lze získat přímočarý výpočet pro hodnotu každého bitu každého kontrolního symbolu, a to postupem uvedeným v následující podkapitole 4.6. Konstantu ze zabezpečovací podmatice tímto postupem můžeme rozvinout do matice binárních prvků velikosti 10×10. Provedeme-li to pro každý prvek matice a tento prvek v zabezpečovací podmatici ní nahradíme, vzniká nová matice velikosti 1290×140 – nyní už ale prvků z GF(2). Matice je xorovou sítí. Určuje každý kontrolní bit operací xor bitů zabezpečované zprávy, na jejichž řádku je v jeho sloupci hodnota 1. Tato implementace vytvořená ve spolupráci s vedoucím práce výrazně zkrátila dobu syntézy na přijatelných 15 minut.

4.6 Optimalizace násobení konstantou v GF(2^r)

Jak lze vidět v kapitole 2.2.1 o konečných tělesech, násobení prvků tělesa $GF(2^r)$ je složitá operace na několik kroků, kdežto součet je přímočará operace. Pro výpočty, kde se objevuje násobení konstantou je tedy vhodné zabývat se zoptimalizováním této operace. Nyní budeme pohlížet na prvek tělesa opět jako na vektor, z definice 2.2.1 víme o distributivním pravidlu a platí tedy, že operaci násobení lze rozepsat tímto způsobem:

$$\mathbf{a}.\mathbf{b} = (a_0, a_1, ..., a_{r-1}).\mathbf{b} = ((a_0, 0, ..., 0) + (0, a_1, 0, ...) + ... + (0, ..., 0, a_{r-1})).\mathbf{b} =$$

$$= (a_0, 0, ..., 0).\mathbf{b} + (0, a_1, 0, ...).\mathbf{b} + ... + (0, ..., 0, a_{r-1}).\mathbf{b}$$

Proto pokud **b** je pevně dané, dává smysl převést násobení konstantou na r-ární součet členů, kde r členů předvypočteme. To se dá použít např. při implementaci logickými hradly v hardwaru, protože nemusíme vyrábět plnou násobičku pro danou aritmetiku, ale vystačíme si s jednodušší násobičkou násobící pevně daným číslem. Matematicky se dá takové řešení zapsat pomocí *násobící matice* velikosti $r \times r$, jejíž řádky jsou předvypočteny, jako:

$$\mathbf{a}.\mathbf{b} = \mathbf{a} \times \begin{bmatrix} 2^0.\mathbf{b} \\ 2^1.\mathbf{b} \\ \vdots \\ 2^{r-1}.\mathbf{b} \end{bmatrix} = \mathbf{a} \times \begin{bmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,r-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r-1,0} & c_{r-1,1} & \cdots & c_{r-1,r-1} \end{bmatrix}$$

Příklad 4.6.1. Optimalizujte součin konstantou **a**.10 násobící maticí nad $GF(16)$ z předchozích příkladů. Vypočtěte ní součin s $a = 13$.

$$\mathbf{a}.\mathbf{b} = \mathbf{a} \times \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{aligned} &1.10 = 10 = (0, 1, 0, 1) \\ &2.10 = x.(x^3 + x) = x^4 + x^2 \bmod G(x) = \\ &= x^2 + x + 1 = 7 = (1, 1, 1, 0) \\ &4.10 = x^2.(x^3 + x) = x^5 + x^3 \bmod G(x) = \\ &= x^3 + x^2 + x = 14 = (0, 1, 1, 1) \\ &8.10 = x^3.(x^3 + x) = x^6 + x^4 \bmod G(x) = \\ &= x^3 + x^2 + x + 1 = 15 = (1, 1, 1, 1) \end{aligned}$$

Nyní vypočteme součin $13 \cdot 10$ a výsledek ověříme podle příkladu 2.2.2:

$$13.10 = 1.(0, 1, 0, 1) + 0.(1, 1, 1, 0) + 1.(0, 1, 1, 1) + 1.(1, 1, 1, 1) = (1, 1, 0, 1) = 11$$

Optimalizovaný výpočet je vlastně jednoduché zapojení hradel $\oplus$ (xor), tzv. *xorová síť*.

4.7 Výsledky

Tabulka 4.2 obsahuje sledované parametry pro porovnání jednotlivých syntetizovaných designů. Syntéza byla provedena ve všech případech nástrojem Vivado 2016.3 pro FPGA XC7VH580T. Byla spouštěna na serverech projektu Liberouter s parametry: 4jádrový procesor Intel Xeon E3-1280 V2 s frekvencí 3,60 GHz, HyperThreadingem (8 vláken) a turbo frekvencí 4,00 GHz; 32 GB RAM a virtualizovaným operační systémem Red Hat Enterprise Linux 5. Syntéza probíhala ve 4 vláknech.

Design #	provedená úprava	doba syntézy	využití LUT	v %	využití registrů	v %	$\sim f_{max}$ v MHz
1	—	6:32	52 089	14,3	5 146	< 1	–
2	rozvinutí násobení	10:01	53 325	14,6	5 151	< 1	–
3	zřetězené zpracování	0:43	15 166	4,2	1 436	$\ll 1$	88
4	vlastní logika	0:17	14 021	3,9	1 436	$\ll 1$	85

Tabulka 4.2: Výsledné parametry jednotlivých designů

Výsledky potvrdily předpoklady, že za dlouhou dobou syntézy stojí jak samotná rozsáhlost logické sítě pro výpočet zabezpečení (rozdíl mezi #1 a #3), tak její vysokoúrovňový popis ve funkcích (rozdíl mezi #3 a #4). Rozdíl mezi posledními dvěma designy je zajímavý také z hlediska maximální frekvence a využitého počtu LUT, neboť se jedná o ekvivalentní obvody – pouze jinak popsané ve VHDL.

Testování

Výsledný design byl prověřen sadou simulačních testů (tzv. *testbench* vytvořený rovněž ve VHDL). Podklady pro základní test – ověření správné funkce kódovacího obvodu jsou obsaženy v příloze normy ([10], příloha 91A). Ta obsahuje 80 po sobě jdoucích a zarovnaných 66b bloků z PCS jako vstup pro transkodér a pro porovnání pak utvořené kódové slovo. V této implementaci je vstupem do transkodéru přímo vstup PCS linek do vrstvy RS-FEC. Test kontroluje především funkci enkodéru, neboť vstup obsahuje pouze samé řídící bloky. Takže nepokrývá dostatečně funkci transkodéru. Proto jsem provedl i další testy zaměřené na zbývající případy chování transkodéru – vstupem jsou samé datové bloky, různě prokládané datové a řídící bloky a také bloky s neplatnými synchronizačními hlavičkami. Design prošel všemi simulovanými testy.

Kapitola 5

RS-FEC vrstva přijímací strany

Jak přijímací, tak vysílací strana jsou si koncepčně podobné, velmi podobný je tedy i jejich návrh. Proto je tato kapitola napsána s odkazem do návrhu vysílací strany a především je napsána jako soupis změn vůči návrhu vysílací strany. Ovšem, s výjimkou dekódovacího obvodu, jež je speciální částí přijímací strany RS-FEC vrstvy.

5.1 Obecný návrh

Podobně jako norma klade požadavky na parametry vysílací strany, klade i následující na přijímací stranu:

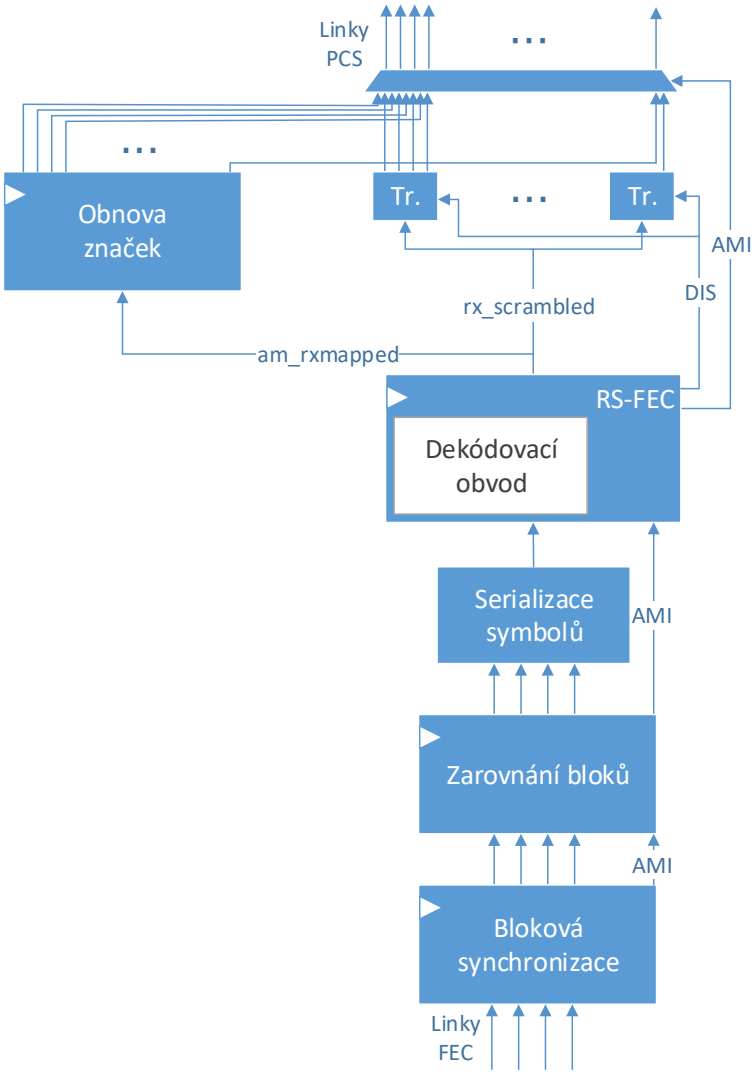
- Schopnost vyrovnat až 180ns relativní posun (skew) přijatých slov mezi linkami FEC
- Propustnost 103,125 Gb/s
- Zpoždění RS-FEC vrstvy vysílací a přijímací cesty dohromady nesmí překročit 409,6 ns[1]

Ačkoliv norma stanovuje maximální zpoždění RS-FEC vrstvy, v reálných situacích požadujeme toto zpoždění co nejnižší. Vzhledem k tomu, že navržená vysílací strana má zpoždění 25,6 ns, je pro přijímací stranu stropem 384 ns, což odpovídá 30 čtvrtinovým částem RS slov. Schéma přijímací strany je na obrázku 5.1, jednotky na něm zobrazené budou nyní popsány.

Bloková synchronizace: Bloková synchronizace probíhá opět na každé lince zvlášť. Na rozdíl od synchronizace ve vysílací straně, zde se na každé lince hledá pouze 64b blok odpovídající AM linky 0. Přičemž ten může být poškozený, porovná se proto po 12 niblech značky (bez paritních oktetů – viz tabulka 4.1) a pokud alespoň 8 niblů odpovídá niblům hledaného AM, předpokládá se, že zbylé byly poškozeny při přenosu a že značka byla nalezena. Tím je nalezen začátek slova RS kódu. Z jedné ze třech následujících 64b značek lze pak zjistit, o jakou FEC linku se jedná, a tuto informaci uložit. Je-li nalezena na odpovídajícím místě po zpracování 4096 slov opět stejná značka, dochází k block locku. Není-li, provádí se slip. Linky na výstupu pro jednotku zarovnání bloků jsou seřazeny tak, aby odpovídaly pořadí odesílatele.

Zarovnání bloků: Funguje stejným způsobem jako funkce zarovnání bloků ve vysílací straně. Rozdílem je, že pracuje namísto 20 linek pouze se 4 linkami a tedy i 4 registry, avšak v úměře k tomu s větší šířkou. Také je hlubší a sice má $180/12,8 \doteq 15$ úrovní.

[1] Tím se myslí Tx a Rx cesty RS-FEC vrstvy téhož zařízení.



Obrázek 5.1: Schéma přijímací strany RS-FEC vrstvy

Symbolová serializace: Serializuje po symbolech data ze seřazených a zarovnaných linek pro jednotku RS-FEC a aplikuje tak reverzní operaci k symbolové distribuci.

RS-FEC: Zajišťuje opravné dekódování pomocí dekódovacího obvodu popsáného podrobně v další podkapitole. Pokud dekódovací obvod označí některé slovo za chybné a neopravitelné, zajistí jednotka záměrné zneplatnění synchronizační hlavičky prvního 66b bloku pro každý lichý 257b blok vyskytující se uvnitř poškozeného slova. Zajistí to po 4 takty pokynem k zneplatnění transkodérům (střídavě lichým a sudým) signálem `DIS` (DIScard).

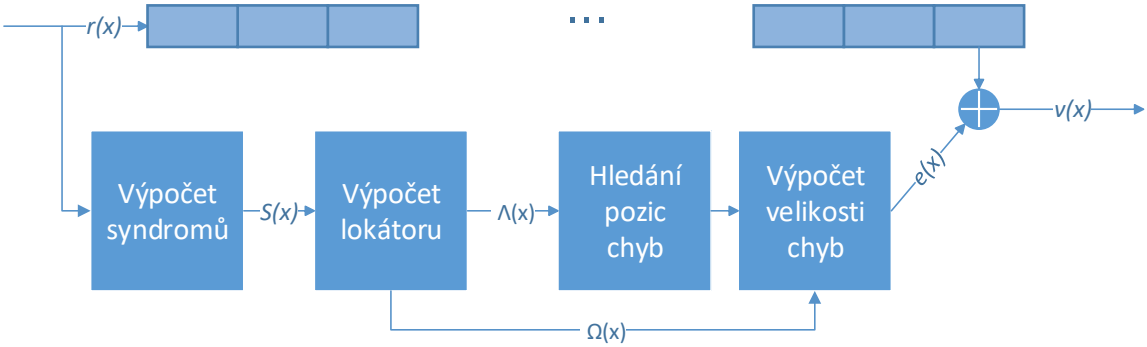
Transkodéry: Zajišťují deskramblování prvních 5 bitů a dále zpětné překódování z 256/257B na 64/66B – čtveřice bloků z po sobě jdoucích linek. Jedná se o opačný proces, než je uvedený v kapitole Tx strany. Avšak jedná se o detranskódování, je tak potřeba dopočítat nibl, který byl v prvním CMD bloku transkódováním vypuštěn (není potřeba pro 257b datové bloky, které se přenáší celistvé). V CMD bloku první dva nibly (po 2bitové hlavičce) označují typ bloku, přičemž

každý typ bloku má unikátní první i druhý nibl. Pomocí vyhledávací tabulky tak je možné zjistit podle jednoho niblu druhý. Protože jsou ale data zaskramblována, je potřeba ponechaný nibl nejprve deskramblovat, vyhledat jemu odpovídající druhý nibl a ten poté zaskramblovat. Každý transkodér tak obsahuje čtveřici 4bitových skramblerů a deskramblerů pro všechny ze 4 možných pozic. Sebesynchronní ethernetový skrambler je dán vazebním mnohočlenem $V(x) = x^{58} + x^{39} + 1$. Popis a schéma skrambleru i deskrambleru jsou uvedeny v kapitole 2.1.1. Pro transkodér prvního 257b bloku je nutné uchovávat dopřednou vazbu (část z posledního 66b bloku z předchozího taktu) pro případ, kdy první CMD blok v 257b bloku je na 1. možné pozici a dopředná vazba „vede“ do části slova z minulého taktu. Proto je jako jediný z transkodérů na obrázku sekvenční.

Obnova značek: Jednotka obnovuje původní tvar zarovnávacích značek tak, aby mohly být poslány na jednotlivé PCS linky. Do značek odpovídajícím linkám 1, 2, 3 a 17, 18, 19 doplní správné hodnoty do oktetů $M_0, M_1, ..., M_6$ (viz tabulka 4.1). Jednotka pracuje paralelně s transkodéry. Výstup na PCS linky řídí multiplexor a bloky z této jednotky odešle pouze při aktivním signálu `AMI`. Ten prochází přes předcházející sekvenční jednotky, které ho náležitě opožďují stejně jako samotná data.

5.2 Návrh dekódovacího obvodu

Dekodér je navrhnut podle kroků výpočtu uvedenými v kapitole 2.2.4. Funkční bloky ve schématu dekodéru na obrázku 5.2 jim odpovídají. Protože by zřejmě bylo velmi náročné provést všechny výpočty vedoucí k opravě chyb v jednom taktu, obsahuje dekodér také zpožďovací registr. To umožňuje zřetězené zpracování po jedné čtvrtině RS slova. V každém taktu do zpožďovacího registru vstoupí jedna tato část přijatého a potencionálně přenosem poškozeného slova označená vstupem $r(x)$. Rovněž každý takt vystoupí opravená jedna část na obrázku označená výstupem $v(x)$. Hloubka registru odpovídá počtu taktů potřebných na výpočet opravy přijatého slova. Zejména na ni závisí zpoždění RS-FEC vrstvy na Rx cestě. Přičemž není možné dosáhnout nižšího zpoždění než 4 takty, neboť syndromy mohou být vypočítány teprve po přijetí celého kódového slova. Zároveň však každé 4 takty přijde jedno slovo, takže propustnost každé jednotky dekodéru musí být 1 slovo za 4 takty.



Obrázek 5.2: Blokové schéma Reed-Solomonova dekodéru

V teoretické části bylo ukázáno, že chybový syndrom S_i daného slova lze spočítat také jako $r(\alpha^i)$. Protože se však každé slovo skládá z 528 symbolů rozdělených na 4 části po 132 symbolech, není potřeba čekat se zahájením výpočtu na příchod celého slova. Slovo můžeme přepsat na součet

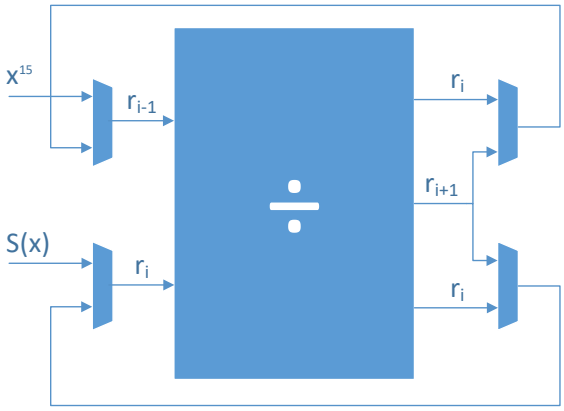
jeho částí 0 až 3 jako:

$$r(x) = r_3(x).x^{3.132} + ... + r_0(x) = ((r_3(x).x^{132} + r_2(x)).x^{132} + r_1(x)).x^{132} + r_0(x)$$

kde každá část je stupně 131. Po dosazení jsou jediné neznámé symboly přijatého slova. Pro každý syndrom tak postačuje 132 násobiček konstantou a sčítaček, které budou použity pro iterační výpočet po jednotlivých částech dle vzorce.

Po získání hodnot syndromu se použije rozšířený Euklidův algoritmus pro hledání NSD x^{15} a $S(x)$ (stupně až 14). Algoritmus končí při získání mezivýsledku stupně menšího než 7. Po každém jeho kroku dojde ke snížení stupně dělence nejméně o 1. Maximálně tak trvá pro tento problém Euklidův algoritmus 8 kroků. Jeden krok Euklida sestává z několika kroků dělení. Pokud však jeho krok nesníží stupeň dělence o více než 1, proběhly nejvýše 2 kroky dělení. Celkově se tedy při řešení tohoto problému provádí maximálně 16 kroků dělení.

Z jednoho kroku dělení vytvoříme HW komponentu a nazveme ji „dělička“. Aby postačilo 16 takových děliček propojených za sebou, je potřeba, aby každá z nich uměla nejen provést samotný krok dělení, ale v případě, že dále už dělit nejde, i připravit další krok Euklida. Takové zapojení lze vidět v iterační podobě na obrázku 5.3. Spodní pozice výstupních multiplexorů představuje další krok dělení, zatímco vrchní přechod na další krok Euklida. Vrchní pozice Euklida u vstupních multiplexorů představuje inicializaci výpočtu, poté zůstává ve spodní pozici. Takováto dělička by výpočet prováděla 16 taktů, proto je třeba zapojit kaskádu alespoň 4 děliček.



Obrázek 5.3: Zapojení děličky pro sériové zpracování

Protože v případě děličky se jedná o dělení s libovolným koeficientem nejvyššího členu dělitele (na rozdíl od problému zakódování, kde je vždy 1), je potřeba předvypočtenou vyhledávací tabulku multiplikativních inverzů všech prvků těles (logaritmická tabulka inverzů). Ta se použije pro převod dělení na násobení inverzem. Tedy pro GF(1024) tabulka o velikosti 1024.10 b $= 10$ kb. Dále se v jednom kroku obecně až $13\times$ násobí (jednotlivé členy dělitele), a to předem neznámou hodnotou, takže každá z děliček vyžaduje navíc 13 úplných násobiček.

Je tak vhodné použít z hlediska zdrojů kaskádu 4 děliček se 4 iteracemi a tedy zpožděním 4 taktů, ale zároveň znovupoužitím tabulky a násobiček, než kaskádu 16 děliček s 1 taktem zpoždění a 3 takty nevyužitou. Obrázek děličky pro přehlednost nezobrazuje výstupy a vstupy proměnných u_i Euklidova algoritmu. Po konečném kroku Euklida zbývá hledání koeficientu c přes inverzní tabulku a vynásobení výsledků jím pomocí úplných násobiček.

Po získání chybového lokátoru se pozice chyb zjistí hrubou silou dosazením všech možných mocnin primitivního polynomu α do něj. Zkoušených pozic a tedy i mocnin je 528, což by zna-

menalo v jednom kroku 528×7 násobiček konstantou. Proto raději bude výpočet opět probíhat iterativně ve 4 taktech, čímž se ušetří polovina násobiček. Označíme-li obecné dosazení do lokátoru touto konvencí:

$$\Lambda_0(x) = \Lambda(x) \quad \Lambda_1(x) = \Lambda(\alpha.x) \quad \Lambda_2(x) = \Lambda(\alpha^2.x) \qquad ... \qquad \Lambda_i(x) = \Lambda(\alpha^i.x)$$

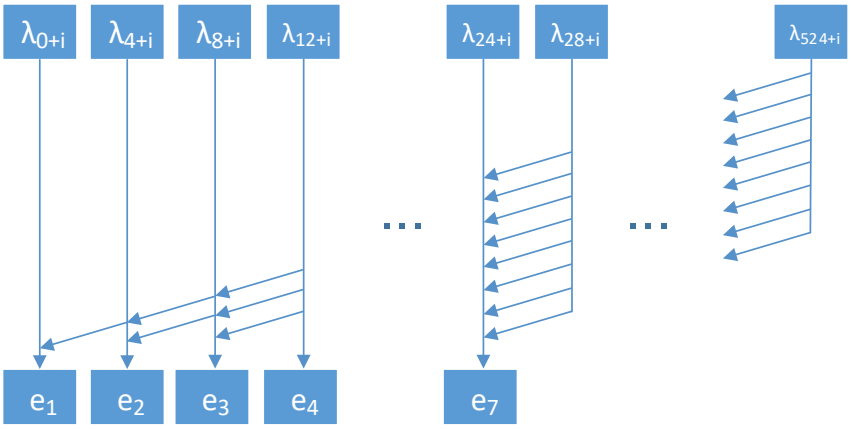
potom platí i:

$$\Lambda_1(x) = \Lambda_0(\alpha.x) = \Lambda(\alpha.x) \quad \Lambda_2(x) = \Lambda_1(\alpha.x) = \Lambda(\alpha^2.x) \quad ... \quad \Lambda_i(x) = \Lambda_{i-1}(\alpha.x) = \Lambda(\alpha^i.x)$$

a označíme-li dosazení $x = 1$ do předchozích rovnicí:

$$\lambda_i = \Lambda_i(1) = \Lambda(\alpha^i)$$

Přičemž $\lambda_i = 0$ znamená že na pozici i se vyskytla chyba. Můžeme tedy výhodně počítat iterativně. V prvním kroku spočteme (inicializujeme) $\Lambda_0(x), \Lambda_4(x), ...$ a $\lambda_0, \lambda_4, ...$ V druhém kroku $\Lambda_1(x), \Lambda_5(x), ...$ a $\lambda_1, \lambda_5, ...$, atdl. Tímto postupem bude postačovat jedna sada násobiček pro inicializaci a jedna pro iterování.



Obrázek 5.4: Multiplexorová síť pro uložení pozic chyb

Přestože najednou vyšetřujeme pouze čtvrtinu možných pozic chyb, můžeme v jednom kroku objevit libovolný počet 0 až 7 chyb. Proto je potřeba prioritní multiplexorovou síť jako je na obrázku 5.4 pro uložení 7 chyb ze 132 potencionálních pozic (horní boxy). Každá šipka představuje multiplexor 1 ze 2, v kterém má přednost chyba na nižším indexu. Pokud na nižším indexu chyba nenastala, propouští se chyba z vyšších indexů. Např. pokud nastala chyba pouze na pozici 524, probublá až do registru e_1. Iterační hledání chyb tak také umožní v každém taktu znovupoužít tuto multiplexorovu síť.

Pro výpočet velikosti chyb je použit Forneyho vzorec. To znamená pro každou 1 ze 7 možných chyb jeden výpočet používající plné násobičky a 1 tabulku inverzů. Hodnoty chyb je potřeba uložit do chybového polynomu, což představuje podobnou multiplexorovu síť jako na obrázku 5.4, pouze s opačnou funkcí. Je proto vhodné implementovat iterační zpracování pro 2 chyby zároveň. Návrh dekodéru pracuje se zpožděním daným:

- 4 takty pro výpočet syndromů
- 1–4 takty pro Euklidový algoritmus, dle míry rozvinutí algoritmu
- 4 takty pro hledání kořenů rovnice lokátoru

- 4 takty pro výpočet hodnoty chyb a sestavení chybového polynomu
- 1 takt pro opravení chyby

Celkově tak zpoždění dekodérem trvá 14–17 taktů dle počtu iterací na výpočet Euklidova algoritmu. To odpovídá 179,2 ns až 217,6 ns. Návrh nepředpokládá, že by bylo realizovatelné snížení odezvy při hledání kořenů, nebo výpočtu chybového polynomu.

Kapitola 6

400Gb/s Ethernet na platformě Xilinx UltraScale

Ačkoliv v době psaní této práce ještě není vydán standard pro 400Gb/s varianty Ethernetu (měl by být na začátku roku 2018), zásadní parametry už jsou veřejnosti známé z dokumentu Baseline summary [14] pracovní komise IEEE. Jsou v něm uvedeny informace o aktuálním stavu vývoje normy a zbývajících cílech. Zde diskutované parametry se už pravděpodobně nezmění. Norma pravděpodobně použije nově 50GHz transceiverů – navýší tak dvojnásobně rychlost dosud nejrychlejších použitých. Počítá se také s použitím kódování *PAM-4* (Pulse Amplitude Modulation), které je používáno i v některých pomalejších variantách. Jedná se o 4stavovou modulaci vysílaného signálu, neboli kvaternární přenos, což v důsledku dvojnásobně zvýší propustnost. Kombinací rychlejšího transceiveru a PAM-4 tak vzniká vysílání o přenosové rychlosti 100 Gb/s, protože jeden symbol nese informaci o 2 bitech. Jednou variantou 400 Gb/s Ethernetu tak bude 4×100 Gb/s, resp. 4×50 GBd. Připravovány jsou ale další dvě varianty a sice 8×50 Gb/s (PAM-4) a 16×25 Gb/s (bez PAM-4). Počítá se s nárůstem výskytu chyb, způsobených zejména nižším rozdílem hodnoty signálu mezi dvěma úrovněmi vyplývajícím ze 4stavové modulace. Proto se považuje za samozřejmé využití RS-FEC nad $GF(2^{10})$ jakožto povinné součásti všech variant. Podle předpokladů navíc stejného kódu, jako je již použit v 100 Gb/s Ethernetu a to buď u varianty KP4 – RS(544, 514), nebo u SR4 – RS(528, 514). Přesné definice RS kódů pro jednotlivé varianty však ještě nejsou známy.

Platforma Virtex-7 z rodiny FPGA čipů 7 Series (popisované v kapitole 3.1) již při psaní této práce nebyla to nejmodernější, co firma Xilinx ve svém portfoliu má. Těmi je rodina čipů UltraScale a UltraScale+, z nichž nejvýkonnějšími jsou čipy Virtex UltraScale(+). [15] Nové čipy obsahují na rozdíl od starší generace 7 Series hardwarově zabudované 100Gb/s ethernetové bloky, nikoliv pouze samotné transceivery. Ty v případě UltraScale+ varianty obsahují i RS-FEC vrstvu. Nová generace čipů Virtex má dále až dvojnásobný počet LUT oproti předchozím [9]. CLB obsahují stejný počet 8 LUT a 16 registrů jako v Series 7, ale velkým rozdílem je, že nyní jsou všechny obsaženy v jediném slice. To zvyšuje možnost routování pro bloky nacházející se navzájem poblíž, neboť ve starší generaci 2 slicy v 1 CLB pracovaly nezávisle a nebylo je možné jednoduše propojit lokálními propojovacími cestami. Další výrazná změna je v jednotkách *DSP* (Digital Signal Processing), jež jsou běžně obsaženy v FPGA a používají se na aritmetické a bitové operace nad bitovými vektory (tj. jedná se o SIMD jednotku – Single Instruction Multiple Data). Z hlediska kódování je zajímavá její funkce xor s 96bitovým vstupem a volbou šířky operace 12–96 bitů. DSP v předchozí generaci umožňuje provádět logické funkce také nad 96 bity, ale pouze se šířkou operace 2 bity. UltraScale obsahuje řádově stejné množství DSP v tisících jako 7 Series. UltraScale+ jich však obsahuje výrazně více a nabízí i variantu s 12 tisíci DSP bloky, tedy ekvivalentem 1,15 milionu operací xor.

Při použití čipů UltraScale pro 100Gb/s Ethernet je výhodné použití HW ethernetových bloků, zatímco na čipech 7 Series musí být implementovány ve firmwaru (v logice FPGA) a zabírají tak nemalé zdroje. Při použití ekvivalentu RS kódování SR4 ve 400Gb/s Ethernetu dojde k navýšení potřebných HW zdrojů až na 4násobek oproti 100Gb/s. Při použití ekvivalentu KP4 až 8násobku, na což Virtex-7 nedostačuje, neboť buď neobsahuje dostatek LUT, nebo neobsahuje potřebné transceivery. Platforma UltraScale se naopak jeví pro 400Gb/s jako vhodná díky většímu počtu LUT i lepším možnostem routování. Ale je možné vyjádřit skepsi nad implementací 400Gb/s s FEC ekvivalentem KP4 na UltraScale (bez +). K tomuto pravděpodobně bude potřeba nejméně UltraScale+ kvůli množství jeho DSP bloků, pakliže pomocí nich bude možno efektivně toto zabezpečení realizovat. Architektura 400 Gb/s totiž neumožní implementaci v UltraScale+ pomocí 100Gb/s ethernetových bloků.

Kapitola 7

Závěr

Cílem práce bylo navrhnout RS-FEC vrstvu pro standard 100Gb/s Ethernetu, zajišťující Reed-Solomonovo samoopravné kódování datového přenosu. Kromě toho vysílací část této vrstvy implementovat v FPGA čipu a zasazením do již hotových ostatních vrstev Ethernetu vytvořit jeho funkční variantu využívající opravné kódování. Jak návrh obou částí, tak vlastní implementace vysílací části jsou hotovy. Implementace je navíc realizována s několika optimalizacemi tak, aby byla použitelná pro reálné nasazení. Byla také otestována na správné chování při simulaci.

Abych mohl vytvořit tento návrh, bylo potřebné nastudovat normu pro 100Gb/s Ethernet. Ta se však nijak podrobně opravným kódováním RS nezaobírá, naopak se pochopitelně zabývá všemi ostatními funkcemi, jež vrstva zajišťuje. Proto bylo navíc nutné nastudovat matematicky založenou problematiku z oblasti kódování a zejména teorii Reed-Solomonových kódů z literatury. Protože Reed-Solomonovy kódy jsou podtřídou cyklických kódů, které jsou podtřídou lineárních blokových kódů, studoval jsem i je. Tyto znalosti jsem zužitkoval při vytváření návrhu opravného (dekódovacího) obvodu pro Reed-Solomonův kód a také pro optimalizace výsledné implementace vysílací strany, resp. jejího kódovacího obvodu.

Prvotní, naivní implementace sice plnila funkční požadavky, ale syntéza tohoto kódu trvala v řádech hodin. Navíc po syntéze spotřebovala přibližně 15 % z celkové počtu LUT na cílovém FPGA. Tam však chceme i v případě využití ethernetového rozhraní (krerého je RS-FEC vrstva pouze jedna část) umístit i akcelerovanou aplikaci. Při syntéze celé takové implementace, včetně aplikace, zřejmě nebudeme chtít tolerovat tak velké prodloužení, zejména používáme-li FPGA pro prototypování. Proto jsem hledal optimalizace a zkoušel je. Některé se ukázaly jako slepá ulička. Avšak aplikací zřetězení na kódovací obvod byla úspěšně snížena náročnost na použité LUT v FPGA přibližně na čtvrtinu. Ke snížení doby syntézy na rozumný čas to však nestačilo. Proto jsme nakonec ve spolupráci s vedoucím práce vytvořili implementaci kódového obvodu napsanou nízkoúrovňově jako síť logických hradel. Zoptimalizovaná implementace spotřebuje pouhých 14 tisíc LUT oproti 55 tisícům prvotní implementace a doba syntézy se zkrátila z 6,5 hodiny na 15 minut.

Díky této práci jsem se dostal ke studiu a implementaci nejmodernějších a nejrychlejších běžně používaný datových přenosů dneška. Také jsem si rozšířil obzory na poli diskrétní matiky i v oblasti programování firmwaru.

Na tuto práci chci navázat implementováním vytvořeného návrhu RS-FEC vrstvy přijímací strany do FPGA, zejména samotného dekódovacího obvodu. Dá se předpokládat, že při tom bude nutné hledat další optimalizace, podobně jako to bylo nutné u kódovacího obvodu. I poté práce může pokračovat dále, a sice rozšířením návrhu a implementace pro ostatní verze Ethernetu využívající RS kódování, zejména rychlosti 200 Gb/s a 400 Gb/s připravovaného standardu.

Literatura

[1] Vodrážka, J.; Pravda, I.: *Principy telekomunikačních systémů*. Česká technika – nakladatelství ČVUT, 2006, ISBN 80-01-03366-X.

[2] Němec, K.: *Datová komunikace*. VUTIUM, 2000, ISBN 80-214-1652-1.

[3] Bidlo, M.: *Návrh počítačových systémů: Kódy pro detekci a opravu chyb*. Brno: FIT VUT v Brně, 2015.

[4] Clarke, C. K. P.: *Read-Solomon error correction*. BBC, 2002, [online] [cit. 1. 7. 2016]. URL http://downloads.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP031.pdf

[5] MacWilliams, F. J.; Sloane, N. J. A.: *The Theory of Error-Correcting Codes*, ročník 16. North-Holland Publishing Company, 1978, ISBN 0-444-85009-0.

[6] Barto, L.; Tůma, J.: *Konečná tělesa*. [online] [cit. 19. 3. 2017]. URL http://www.karlin.mff.cuni.cz/~barto/student/SkriptaKonTel.pdf

[7] Lee, L. H. C.: *Error-Control Block Codes for Communications Engineers*. Artech House, 2000, ISBN 1-58053-032-X.

[8] *7 Series FPGAs Configurable Logic Block*. Xilinx, Inc., 27. 9. 2016 (verze 1.8), [online] [cit. 15. 4. 2017]. URL https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[9] *7 Series FPGAs Data Sheet*. Xilinx, Inc., 28. 3. 2017 (verze 2.4), [online] [cit. 15. 4. 2017]. URL https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[10] IEEE Std 802.3-2015. *IEEE Standard for Ethernet: SECTION SIX*. Standard, IEEE Computer Society, Březen 2016, ISBN 978-1-5044-0078-7.

[11] Wikipedia: Ethernet – Wikipedia, The Free Encyclopedia. 2017, [online] [cit. 18. 4. 2017]. URL https://en.wikipedia.org/wiki/Ethernet

[12] Huff, L.: *High-Speed Transmission Update: 200G/400G*. ConnectorSupplier.com, 18. 6. 2016, [online] [cit. 15. 4. 2017]. URL http://www.connectorsupplier.com/high-speed-transmission-update-200g400g/

[13] *7 Series FPGAs Clocking Resources*. Xilinx, Inc., 1. 3. 2017 (verze 1.13), [online] [cit. 25. 4. 2017].
URL https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

[14] IEEE P802.3bs Baseline Summary. IEEE P802.3bs 400 GbE Task Force, 18. 6. 2015, [online] [cit. 16. 4. 2017].
URL http://www.ieee802.org/3/bs/baseline_3bs_0715.pdf

[15] *UltraScale Architecture and Product Data Sheet*. Xilinx, Inc., 15. 2. 2017 (verze 2.11), [online] [cit. 16. 4. 2017].
URL https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

Příloha A

Obsah přiloženého CD

```
│
├── implementation/ ..... implementace Tx strany RS-FEC vrstvy
│   ├── sim/ ..... simulační testy
│   ├── synth/ ..... projekt Vivada pro syntézu
│   └── *.vhdl ..... soubory implementace a jejich verze
├── text/ ..... zdrojové soubory tohoto dokumentu pro systém LaTeX
├── readme.txt ..... podrobnější informace o obsahu média
└── xvelec07-rs-fec-ethernet.pdf ..... tento dokument
```