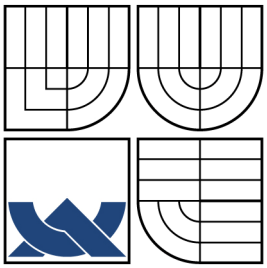
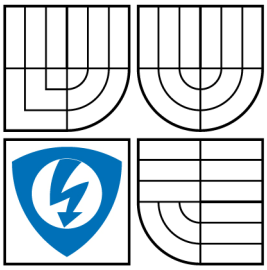


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉMY REALIZACE PROTICHYBOVÉHO KÓDOVÁNÍ

SYSTEMS DESIGN OF CORRECTION CODING

DOKTORSKÁ PRÁCE
DOCTORAL THESIS

AUTOR PRÁCE — Ing. VÍTĚZSLAV KŘIVÁNEK
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. KAREL NĚMEC, CSc.
SUPERVISOR

BRNO 2008

Anotace

Nejen při přenosu dat díky zvyšování přenosové rychlosti se lze stále častěji setkat s chybami, které mají tendenci se shlukovat. Hledání alternativních řešení korekce shlukových chyb k dosud používaným metodám je v oblasti zájmu předkládané práce. Cílem je detailně rozebrat problematiku konvolučních kódů pro opravu shluku chyb, jež mohou být využitelné v individuálních protichybových systémech, a dosáhnout tak lepších výsledků než při hromadné aplikaci stávajících řešení.

Nejprve jsou stručně popsány stávající využívané metody k odstranění či potlačení shlukových chyb. Následuje část věnovaná jednotlivým systematickým konvolučním kódům, kde jsou dané kódy podrobně popsány rozsáhlým matematickým aparátem, jenž rozšiřuje soubor možných hodnotících kritérií protichybových systémů, které jdou uplatnit při posuzování návrhu individuálních řešení. Získané vlastnosti kódů jsou konfrontovány jak mezi konvolučními kódy, tak i s dalšími variantami návrhů ochrany zprávy před shlukem chyb. Pro prověření správnosti odvozeného matematického aparátu jsou zpracované konvoluční kódy podrobeny kontrole pomocí simulací v Matlabu. Jelikož simulace představuje základní používanou metodu pro ověření i prezentování již navrženého zabezpečovacího procesu a umožňuje lepší pohled na danou problematiku. Realizovatelnost individuálních protichybových systémů je následně ověřována pomocí vytváření popisu chování obvodu jazykem VHDL. Jeho velká přenositelnost představuje podstatnou výhodu při návrhu individuálních systémů vlastní realizace.

Klíčová slova:

Protichybové kódování, shlukové chyby, výběrová kritéria, konvoluční kódy, individuální protichybové systémy, počítačová simulace, Matlab, VHDL.

Abstract

Due to growing transmission speed burst-forming errors tend to occur still more frequently not exclusively in data transmission. The presented paper concentrates on the search for alternative burst error correction solutions complementing the existing methods in use. Its objective is an elaboration of a detailed analysis of the issue of convolution codes for error burst correction which can be used in individual anti-error systems and thus an achievement of better results than those attained by mass application of the existing solutions.

First the methods implemented to remove or suppress burst errors are briefly characterized. This part is followed by a detailed description of the individual systematic convolution codes by means of mathematical tools which extend the set of possible evaluative criteria of anti-error systems which can be applied while assessing proposals for individual solutions. The acquired code properties are compared with convolution codes as well as with other versions of proposals for message protection against an error burst. The processed convolution codes are subject to testing by means of Matlab mathematical programme simulation in order to validate the correctness of the derived mathematical tools. This is because simulation represents the principal method applied to verify and present an already proposed security process and enables the acquisition of a better overview of the issue at hand. The feasibility of the individual anti-error systems is then confirmed by way of creating a circuit behaviour description in the VHDL language. Its high portability presents a big advantage when drafting individual systems of the actual implementation.

Keywords:

Forward Error Correction (FEC), Burst-Error, Selection Criterion, Convolution Coding, Individual Anti-Error Systems, Computer Simulation, Matlab, VHDL.

Poděkování

Děkuji mému školiteli Doc. Ing. Karlovi Němcovi, CSc. za velmi užitečnou pedagogickou i odbornou pomoc a cenné rady při zpracování předkládané práce. Taktéž děkuji mé rodině za podporu a vytvoření prostředí pro věnování se vědě.

V Brně dne 21. 8. 2008 ……..

Vítězslav Křivánek

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek a symbolů

A	Ochranný interval
ALU	Arithmetic Logic Unit
AND	Logický součin
b	Shluk chyb
$\mathbf{B_0}$	Vytvářecí bloková matice v binárním tvaru
BCH	Bose-Chaudhuri-Hocquenghemovy kódy
$\mathbf{B_D}$	Vytvářecí bloková matice v dekadickém tvaru
BER	Bit Error Rate
CMOS	Complementary Metal-Oxide-Semiconductor
d	Hammingova vzdálenost
D_{BI}	Zpoždění způsobené prokládací maticí u blokového prokládání
D_{CI}	Zpoždění způsobené prokládací maticí u konvolučního prokládání
DFT	Diskrétní Fourierova transformace
ECL	Emitter Coupled Logic
EEPLD	Electrically Erasable Programmable Logic Device
EEPROM	Electrically Erasable Programmable Read-Only Memory
F	Matice dílčích výstupních toků
$f(i)$	Funkce udávající šířku registrů
f_{IN}, f_{OUT}, f_{SYN}	Frekvence
FPGA	Field Programmable Gate Arrays
G	Vytvářecí matice
GAL	Generic Array Logic
GF	Galois Field
H	Kontrolní matice
HDL	Hardware Description Language
IDFT	Inverzní diskrétní Fourierova transformace
IEEE	Institute of Electrical and Electronics Engineers
IIL	Integratet Injection Logic
ITU	Internacional Telecommunication Union
j	Prokládací faktor
K	Kódové ohraničení
k, k_0	Vstupní dílčí tok
KJ	Komunikační jednotka
KZD	Koncové zařízení přenosu dat
$L(n)$	Počet míst dekadického čísla vyjádřeného binárně
m	Počet bitů na symbol
MAP	Algoritmus Maximum a-posteriori
m_c	Počet paměťových buněk kodéru
n, n_0	Výstupní dílčí tok
NAND	Negovaný logický součin
NPS	Nadřazený přenosový systém

NRC	Nerekurzivní konvoluční kodér
OR	Logický součet
P	Matice dílčí vstupních toků
PAL	Programable Array Logic
PKS	Protichybový kódový systém
PLD	Programmable Logic Device
PLL	Phase Lock Loop
PM	Velikost prokládací matice
PZ	Periferní zařízení
R	Informační rychlost
r	Počet zabezpečovaných symbolů
RAM	Random-Access Memory
RS	Reed-Solomonovy kódy
RSC	Rekurzivní systematický konvoluční kodér
s	Syndromový prvek
SD	Spotřebič dat
S_L	Složitost kodeku – počet logických operátorů
SNR	Signal to Noise Ratio
SOVA	Soft-output Viterbi algoritmus
S_P	Složitost kodeku – počet paměťových buněk
SPD	Systém přenosu dar
SRAM	Static Random-Access Memory
S_x	Rozhraní mezi systémy
t	Počet opravitelných chyb (symbolů)
t_B	Potřebná doba pro zpracování jednoho bloku bitů
T_{IN}, T_{OUT}, T_{SYN}	Doba periody dělené synchronizace
TTL	Tranzistor-Tranzistor-Logic
u_k	Vstupní nezabezpečená posloupnost u Turbo kódů
UZD	Přenosové zařízení přenosu dat
v	Přenosová rychlost
VCC	Vývod napájecího napětí pro jádro obvodu
VCCIO	Vývod napájecího napětí pro vstupně-výstupní bloky
VCCPLL	Vývod napájecího napětí pro fázové závěsy
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access
XOR	Exklusivní disjunkce
y_k	Výstupní zabezpečená posloupnost u Turbo kódů
Z	Zpoždění kodeku
ZD	Zdroj dat

1 Úvod

Zvláště v posledních letech roste potřeba efektivních a spolehlivých systémů přenosu dat. Hlavní příčinou je velmi rychlý rozmach vysokorychlostních systémů pro výměnu, zpracování a uchování dat. S tím souvisí i stálá expanze multimédií a internetu, jež přináší stále větší potřebu i požadavky na používané technologie a jejich vývoj. Pro návrh těchto systémů je zapotřebí slučování komunikačních a počítačových technologií.

Jeden z hlavních zájmů spočívá v kontrole chyb a v následném obnovení získaných dat. Jelikož pro příjemce zprávy má význam pouze bezchybná zpráva, je zřejmá snaha přenášenou zprávu proti chybám zabezpečit. S rozvojem digitálních sítí nabývá téma zabezpečování přenášené informace na významu. Dnes se již standardně a zcela systematicky používají při přenosu zpráv v digitálních sítích bezpečnostní kódy. Děje se tak na základě vkládání další nadbytečné informace pomocí kodéru. Umělé zvyšování nadbytečnosti přenášené posloupnosti signálových prvků ovšem snižuje množství přenášené užitečné informace, avšak dosahuje se potřebného kódového zabezpečení dat pro přenos. Účelem tohoto kroku je případná náprava způsobené chyby na straně přijímače při přenosu dat reálným kanálem.

Fyzikální prostředí, ve kterém je informace přenášena, se nazývá kanál. Příkladem kanálů mohou být atmosféra, telefonní linky, elektromagnetické pole, atd. Při přenosu informace kanálem může dojít k jejímu poškození kvůli poruchám fyzikálního prostředí, ve kterém k přenosu dochází. Obecně se nazývají šum. Šum může být způsobený například slunečními skvrnami, bleskem, přehnutím magnetické pásky, meteorickým rojem, přeslechem u telefonních linek, překlepy při psaní, špatnou artikulací, nedoslýchavostí, poškrábáním kompaktního disku, atd. Šum může způsobit, že přijatá zpráva se liší od zprávy vyslané.

Systém komunikace mezi počítačem a hromadnou pamětí, ať se jedná o optický nebo magnetický záznam, je další oblastí, kde moderní metody kódování tvoří nepostradatelnou součást použití. Za komunikační kanály tedy lze považovat i každé propojení výpočetního systému s hromadnou pamětí. Je to způsobeno především tím, že samotné paměťové médium nebo cesta signálu, jenž zprostředkovává přenos informace k tomuto médiu, jsou prostředími, která podléhají vlivům šumu. Všude tam, kde je nutné eliminovat vliv šumu, který je způsoben okolním prostředím, se uplatňují zabezpečovací kódy.

Též do užšího centra pozornosti souvisejícím s prudkým rozvojem aplikací digitálních sítí se dostává citlivý aspekt nutnosti utajení privátních informací. Řeší se tak požadavek diferenciace přístupových práv jednotlivých uživatelů, který je vhodně prováděn pomocí kryptografických metod založených na kódovacích technikách. Ovšem šifrované zprávy jsou velmi vnímavé na vliv různých šumů. Proto i šifrování má značnou spojitost s použitím vhodných zabezpečovacích kódů [3].

Na základě kódování správně a účinně fungují moderní komunikační prostředky, jakými jsou optické paměti počítačů, družicové spoje nebo mobilní telefony. Zájem většiny výrobců se přirozeně soustřeďuje na dosažení uspokojivě velké spolehlivosti, jakou předepisují pro provoz digitálních sítí příslušná doporučení ITU (Internacional Telecommunication Union).

V průběhu přenosu zprávy může být použito několika druhů signálů k dokončení daného přenosu. Vzhledem k různorodým přenosovým prostředím je vhodné, aby použité zabezpečovací kódování bylo danému prostředí adekvátně přizpůsobeno. V dnešní době je velmi důležité najít co nejvhodnější řešení, které se liší podle nároků na něj kladených, jelikož zabezpečení se dosahuje zvýšením nadbytečnosti na úkor vlastního přenosu informace. Záleží především na reálných podmínkách přenosu od vysílače k přijímači zabezpečované informace.

V současnosti, jistě i do budoucna, je snaha u moderních komunikačních technologií neustále zrychlovat přenos dat. To s sebou přináší nové problémy, které při takovém přenosu vznikají, a je třeba jejich odstranění. Především se při přenosu dat lze díky zvyšování přenosové rychlosti stále častěji setkat s chybami, které mají tendenci shlukovat se. Jedním z hlavních problémů je i úkol najít adekvátní kódování a opravu chyb. Hledání alternativních řešení korekce shlukových chyb k dosud používaným metodám je v oblasti zájmu disertační práce.

Základním cílem práce je nalézt adekvátní náhradu hromadně používaných systémů pro opravu shluků chyb, jejichž nasazení v individuálních protichybových systémech umožní efektivnější přenos dat. Především odvodit podrobný matematický aparát pro rozšíření souboru kritérií, jenž umožní lepší srovnání protichybových systémů vhodných pro přenos digitálních signálů. Simulacemi zvolených systémů ověřit získané teoretické výsledky. Následně výsledky využít k hledání přiměřených realizačních metod s ohledem na umístění do nadřazených přenosových systémů. Vzhledem k další využitelnosti získaných výsledků, zefektivňování procesu návrhu i realizaci číslicových systémů a snadné přenositelnosti je realizace demonstrovaná pomocí programově logických obvodů, jelikož i díky rekonfigurovatelnosti se v dnešní době hardware navrhuje i technikami dříve určenými jen pro návrh software.

2 Možnosti korekce shlukových chyb

Přenos zpráv, který je na vstupu i výstupu příslušného přenosového systému nejčastěji dvoustavový, se v poslední době provádí převážně pomocí diskrétního signálu. Posloupnost signálových prvků představující přenášenou zprávu využívá tedy pro dvoustavový diskrétní signál logické hodnoty 0 a 1. Ve zprávách mohou během přenosu, zejména vlivem aditivního rušení, úniku a mezisymbolových přeslechů, vzniknout chyby. Ty se projeví inverzní hodnotou signálového prvku. Prvek logické hodnoty 0 se změní na 1 a naopak.

Nebezpečí vzniklé chyby pro příjemce představuje především možnost nesprávné reakce systému na přenesenou zprávu. Zmíněný jev nastává tehdy, jestliže se informace přenášená sledovaným úsekem významově změní v jinou a zcela odlišnou část užívané zprávy. Stává se tak, když soubor možných kombinací signálových prvků v dané části zprávy je využíván beze zbytku jen pro přenos informace. Počet míst, ve kterých se dvě kombinace prvků ve sledovaném úseku zprávy mezi sebou liší se nazývá Hammingova vzdálenost *d*.

Umělým zvyšováním nadbytečnosti přenášené posloupnosti signálových prvků se zvyšuje Hammingova vzdálenost, ovšem zároveň se snižuje hodnota přenášené informace. Tímto způsobem lze dosáhnout kódového zabezpečení. Základní myšlenkou korekčních kódů je doplnění původní informace zvláštní kontrolní skupinou oproti původní posloupnosti, jenž není zabezpečená a chráněná proti chybám. Rozšířenou posloupnost použije dekodér korekčního kódu při pokusu zrekonstruovat původní vyslanou informaci.

Stupeň ochrany bývá různý. Čím je provedena lepší ochrana proti chybám, tím více se zvýší požadovaná přenosová rychlost a nebo se zvýší výpočetní náročnost [37]. Podle příslušného použitého systému mohou být zabezpečeny nejen jednotlivé bity, ale i celé byty – paritní kódy; konvoluční kódy; blokový Fireův kód, Reed–Solomonův kód. U složitějších systémů bývá nedílnou součástí kanálového kódování tzv. prokládání (interleaving), který pomáhá zabezpečit signál proti delším shlukům chyb.

Shluky chyb vznikají jako relativně krátké úseky bitového toku, ve kterém jsou jednotlivé chybné bity odděleny velmi malým počtem bezchybných bitů - jsou to úseky s vysokou hustotou chyb. Mezi těmito úseky se nacházejí úseky s nízkou hustotou chyb. Pro korekci uvedených poruch nejsou příliš vhodné systémy pro opravu náhodných chyb, ale používají se systémy pro opravu shlukových chyb. Typickým příkladem, jenž ovlivňuje několik sousedních signálových prvků přenášených komunikačním kanálem může být výboj blesku nebo průmyslové rušení. V současnosti se dále shlukové chyby vyskytují zejména v souvislosti s implementací optických médií [41], kde je použito malých rozměrů uspořádání, či ve vysokorychlostních systémech, kde i krátkodobé rušení představuje rozsáhlejší poškození.

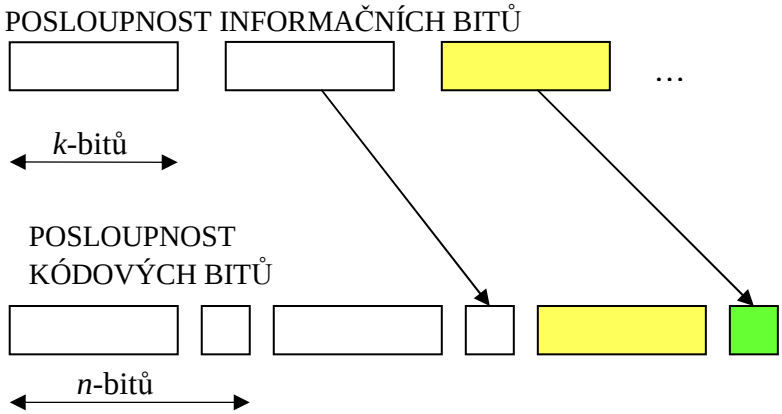
Miniaturizace místa pro záznam dat a zrychlování přenosu dat neustále pokračuje v překotném tempu. Pro spolehlivou funkčnost je nepostradatelné se zabývat ochrannou před nežádoucími vlivy i tam, kde to dříve nebylo nutné [15]. Stále častěji se místo jednoduchých chyb či vícenásobných chyb vyskytují chyby shlukové. K jejich odstranění se používají odlišnější metody [6].

Každoročně se zvyšují objemy přenesených dat. Následkem je velmi rychlé přenášení informace s minimálním časem kladeným na přenos dat mezi vysílačem a přijímačem. Bohužel se při přenosu začínají vyskytovat chyby, které se shlukují do jednoho celku a negativně ovlivňují výkonnost systémů [4]. Pro běžné zabezpečení se stále častěji používají složitější zabezpečovací techniky, což má za následek stále náročnější systémy, které potřebují velmi výkonnou techniku na výpočetní operace a tím dosahují nárůstu délky zabezpečení.

K potlačení shluků chyb či k opravě chybných bitů lze využít několika rozdílných metod. Stručná charakteristika a principy jsou představeny v následujících podkapitolách.

2.1 Blokové kódy

Uskutečňují zabezpečovací proces pouze v rámci jediného bloku, který vznikl oddělením určitého úseku signálových prvků viz. Obr. 2.1. Sledovaný úsek zprávy se nazývá kódové slovo. Při používání blokového kódování velmi rychle narůstá množství redundantních dat s počtem bitů, které má být kódování schopno opravit v daném bloku. Proto se tato metoda využívá zejména tam, kde nárůst datového toku není kritický. Především je uvedená skupina kódů zaměřena na opravu nezávislých chyb.



Obr. 2.1: Princip kódování blokovými kódy.

Pro podrobnější třídění blokových kódů se používá popis uložení vzniklé zabezpečovací nadbytečnosti ve sledované části přenášených signálových prvků (systematičnost, lineárnost). Do skupiny blokových kódů patří například paritní kódy, Hammingovy kódy, Fireovy kódy, Reed-Müllerovy kódy, Bose-Chaudhuri-Hocquenghemovy kódy, Reed-Solomonovy kódy.

2.1.1 Reed-Solomonův kód

Výjimku mezi blokovými kódy ve schopnosti opravování chyb tvoří Reed-Solomonovy kódy, jenž patří mezi nebinární cyklické BCH kódy (Bose-Chaudhuri-Hocquenghem) a jejich abeceda zdroje je vždy vyšší než binární, proto se využívají pro korekci shlukových chyb. Označují se zkratkou RS *(n, k)*, která charakterizuje daný kód. Parametr *k* určuje počet *m* - bitových symbolů vstupujících do kodéru, parametr *n* udává velikost zprávy vystupující z kodéru. To znamená, že počet paritních symbolů v jednom bloku je *n* - *k*. Reed-Solomonův dekodér je schopen opravit maximálně *t* chybných symbolů. Reed-Solomonův kodér tedy na vysílací straně připojí *k* blokům digitálních dat redundantní bity, pomocí nichž dekodér na přijímací straně opraví vzniklé chyby a tím získá původní data. Pro délku jejich slova platí následující vztah:

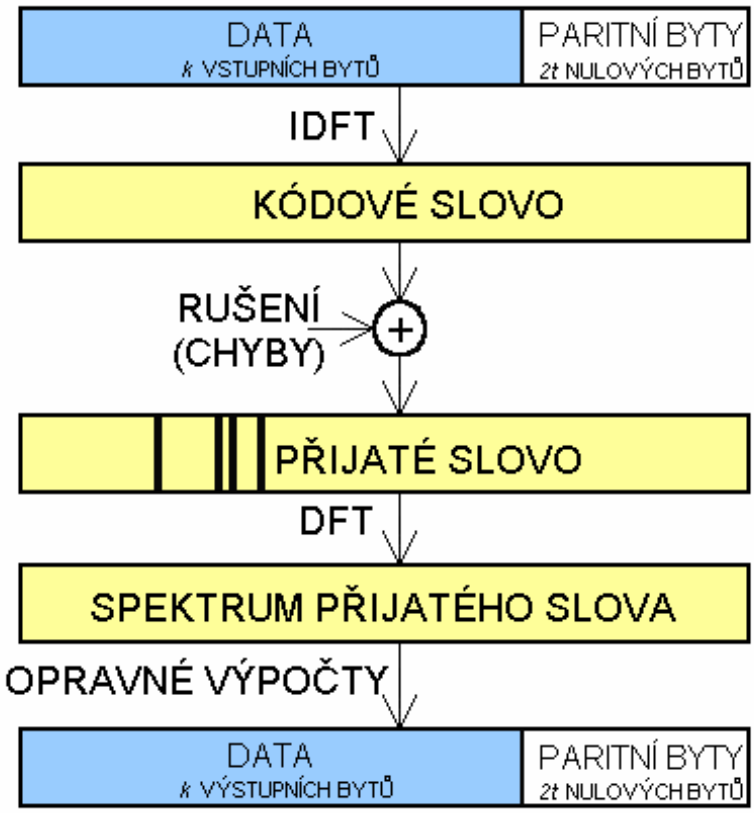
$$\begin{aligned} &2t \le n-k\,, \\ &n = 2^m - 1, \end{aligned} \tag{2.1}$$

kde *n* je délka kódového slova a *m* počet bitů na symbol. Z nebinárního kódu délky $n = 2^m - 1$ se obdrží binární kód délky $n \times m$ tak, že se nahradí každý nebinární symbol jeho odpovídající binární *n*-tic. RS kódy existují pro všechny hodnoty *n*, *k*, pro které platí:

$$0 < k < n < 2^m\,. \tag{2.2}$$

Kódování se neprovádí nad jednotlivými bity, ale nad symboly (byty). Nezáleží na tom, kolik chybných bitů obsahuje jeden symbol, ale pouze na tom, zda je chybný či nikoliv. Uvedené kódy nejsou efektivní pro opravu nezávislých, ojedinělých chyb [33]. Tuto nevýhodu RS kódů lze odstranit pomocí zřetězených kódů, jenž využívají dva kodéry zapojené v kaskádě [12], [34].

Nevýhodou je poměrně složitý proces kódování a dekódování signálu RS kódem [43]. Zjednodušený postup je demonstrován na Obr. 2.2. Vstupní signál RS kodéru se zpracovává postupně po slovech, z nichž každé je složeno z *k* bytů. Ke každému slovu je přidáno 2*t* nulových bytů a získá se výsledné slovo, které má velikost *k* + 2*t* bytů, jenž představuje reprezentaci vstupního signálu v kmitočtové (spektrální) oblasti. Z něj se určí tvar tzv. polynomu Galoisova pole - GF (Galois Field), kde vytvořené koeficienty jednotlivých členů polynomu představují hodnoty jednotlivých bytů výsledného slova. Nad tímto polynomem se následně provádí inverzní diskrétní Fourierova transformace IDFT (signál je transformován do časové oblasti). Výsledkem transformace je kódové slovo u něhož již nelze oddělit vstupní a kontrolní byty, neboť jsou transformací promíchány. Komunikačním kanálem je signál přenášen v této podobě.



Obr. 2.2: Základní postup při kódování a dekódování RS (kódem).

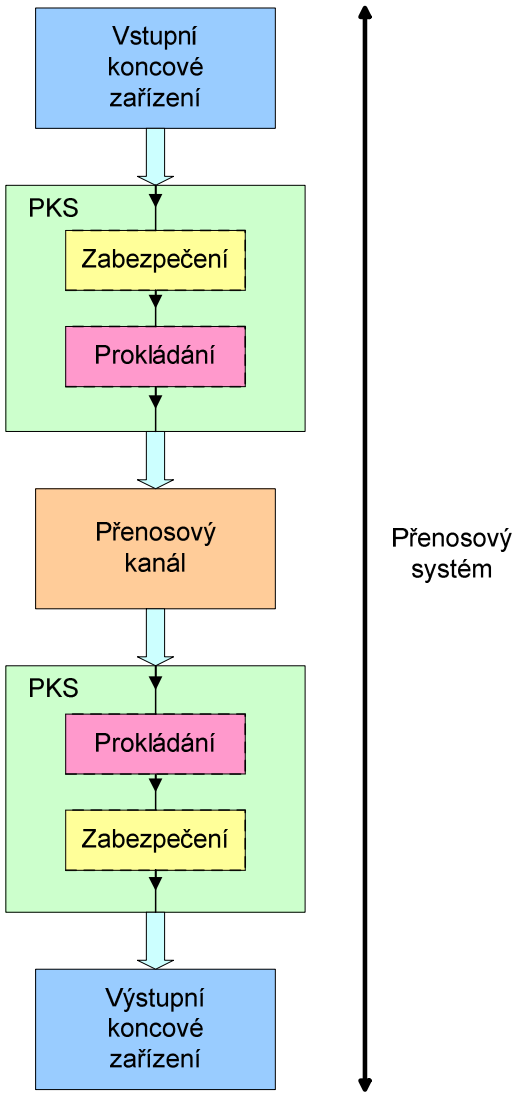
Přijaté slovo se na přijímací straně transformuje přímou diskrétní Fourierovou transformací DFT zpět do spektrální oblasti. Jestliže na přenosové cestě nedošlo k poškození signálu v důsledku rušení, je ve spektru přijatého slova posledních 2*t* bytů nulových a prvních *k* bytů představuje bezchybně přenesený vstupní signál. Jiná situace nastává pokud se signál na přenosové cestě poškodí, tj. vzniknou v něm chyby - na Obr. **2.2** je znázorněno pruhy. Po transformaci DFT již nebude ve spektru přijatého signálu posledních 2*t* bytů nulových. Ovšem při znalosti rozložení jedniček a nul u posledních 2*t* bytů, lze opravnými výpočty s pomocí tzv. lokalizačního polynomu opravit až *t* chybných bytů. Jeho kořeny určují místa chyb v přenášené zprávě. Tato fáze může být řešena více způsoby, např. Euklidovým algoritmem, Berlekamp-Masseyovým algoritmem, atd. [10]. Provedením opravných výpočtů se opět dosáhne stavu, že posledních 2*t* bytů je nulových a prvních *k* bytů představuje opravený původní vstupní signál, pokud nebyla překročena zabezpečující schopnost kódu.

Donedávna byla softwarová realizace v reálném čase vzhledem ke složitosti výpočtů neuskutečnitelná. Aritmetické operace v konečných polích GF(2^m) jsou dosti odlišné od operací v binárním číselném systému. Komplikovaná je zejména realizace násobení a dělení. Tedy velké obtíže u zmíněné implementace jsou hlavně z důvodu nepodporování Galoisova tělesa a k němu patřících aritmetických operací procesorem. Nicméně dnes dostupné výkonné procesory již umožňují zpracování dat velmi vysokými rychlostmi. Avšak stále složitost a výpočetní náročnost kódování je velmi

vysoká, proto je mnohdy vhodnější používat systémy opravující menší shluky chyb, za to však s rychlejším kódováním přenášených dat.

2.2 Prokládání

Tvoří používaný doplněk kanálového kódování při předchozím využití jiného kódování proti chybám [8], [23]. Prokládání (interleaving) se používá jako ochrana signálu proti skupinovým chybám – shluku chyb. Prokládání zprávy představuje přídavný systém, který se vkládá ve vysílači mezi kodér a přenosový kanál a na straně příjemce mezi přenosový kanál a dekodér – naznačeno na Obr. 2.3. Základní princip spočívá ve změně polohy bitových toků tak, aby se charakter distribuce shluku chyb změnil na distribuci nezávislých chyb, jenž lze účinněji odstranit či potlačit. Nevýhoda této dodatečně prováděné operace spočívá v dalším zpožďování dekódování bitů na straně příjemce. Zpoždění způsobují prokládací matice a je určeno jejich velikostí jak v části vysílače tak v části přijímače.



Obr. 2.3: Umístění systému prokládání v přenosovém systému.

Vychází se z předpokladu dvoustavového signálu, který je reprezentovaný bitovou posloupností. Prokládání probíhá vytvořením matice (vstupní paměť), do které se zapisují data po řádcích. Po naplnění celé matice, se provede čtení z matice, ale nikoliv po řádcích, avšak po sloupcích. Obdobný systém je realizován v přijímači, kde se data zapisují do matice po sloupcích a vyčítají z ní po řádcích. V závislosti na počtu řádků a sloupců matice, případně zpoždění lze docílit efektu, změny časové polohy bitového toku a tím potlačení shluků chyb. Avšak za cenu zvýšení zpoždění celého systému.

2.2.1 Bloková metoda prokládání

Do kodéru blokového korekčního kódu vstupuje tok nezabezpečených bitů, kde se k nim přidá ($n - k$) zabezpečovacích bitů tak, aby bylo možno v dekodéru opravit až t chyb. Kódové kombinace jsou ukládány do paměti matice prokládání, jenž má rozměr n x j tak, že v každém řádku je jedna kódová kombinace. Až se matice určená pro prokládání naplní, jsou z ní uložené bity vysílány do přenosového kanálu, ale tentokrát po jednotlivých sloupcích. Vzniklý bitový tok je přenášen přes přenosový kanál do stejné paměťové matice, kde je ukládán po sloupcích. Po naplnění paměťové matice jsou přenesené bity vysílány po řádcích do dekodéru příslušného korekčního kódu, kde jsou chybně přenesené bity opraveny. Z uvedeného popisu je zřejmé, že ve sdělovacím kanále jsou každé dva bity původní kódové kombinace odděleny ($j - 1$) bity ostatních kódových kombinací. Je-li pak přenášený bitový tok napaden shlukem chyb, je vzhledem k použitému korekčnímu kódu napadeno všech j kódových kombinací, avšak j-krát menším počtem chyb.

Parametry, které charakterizují daný protichybový systém, jsou tvořeny parametry vlastního blokového kódu, tj. (n, k) a tzv. prokládacím faktorem j, který udává počet kódových kombinací, jenž mají být proloženy. Pro stanovení rozměrů prokládací paměťové matice je třeba znát maximální statisticky významnou délku shluků chyb b v bitech a minimální délku bezchybného intervalu A v bitech mezi jednotlivými shluky chyb, které byly nalezeny při statistickém rozboru distribuce chyb v použitém kanále. Pak rozměry prokládací paměťové matice musí vyhovovat nerovnosti:

$$j \geq \frac{b}{t}, \tag{2.3}$$

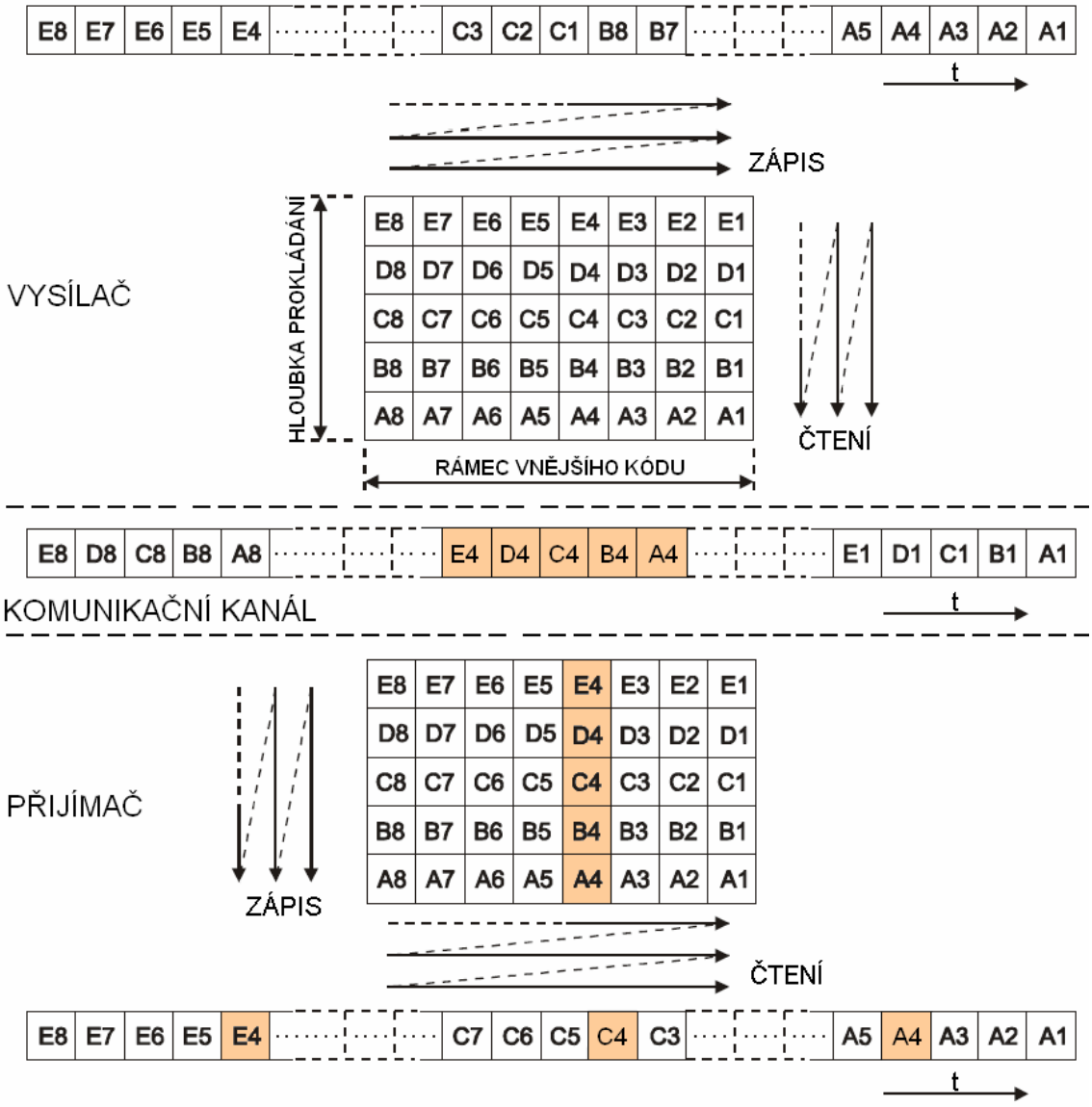
kde t je počet nezávislých chyb opravovaných kódem. U kódů, které korigují shluky chyb se místo t používá délka shluku b. Pro stanovení počtu sloupců n je třeba splnit nerovnost:

$$n \leq \frac{A+b}{j}. \tag{2.4}$$

Zpoždění D_{BI} vyplývající z prokládání bitů, lze odvodit z rozměrů prokládacích matic (*PM*), pomocí vzorce:

$$D_{BI} = 2 * PM = 2 * j * n \,. \tag{2.5}$$

Pro minimalizaci zpoždění je při prokládání bitů zapotřebí použít paralelních pamětí. Důvodem je, aby se nemusel zastavit tok dat a počkat, až se vynulují jednotlivé matice, jelikož po naplnění matice má probíhat čtení z matice na straně vysílače či na straně přijímače, případně další plnění matice. Proto se z technických důvodů zajišťují větší kapacity pamětí, jelikož když se jedna matice naplní, tak další matice přebírá funkci na zápis a obráceně [8].



Obr. 2.4: Základní princip blokového prokládání.

Názorný příklad principu je uveden na Obr. 2.4. Vstupní signály bitového toku s pořadím bitů A1, A2, A3, ... E6, E7, E8 se ve vysílací části ukládají do paměti po řádcích. Avšak vyčítání bitů se z paměti provádí po sloupcích. Ve srovnání se vstupním signálem má signál na výstupu paměti pořadí bitů změněno. Komunikačním kanálem se přenáší bitový tok s pořadím bitů A1, B1, C1, ... C8, D8, E8. Na přijímací straně je proveden inverzní postup. Tedy přijatý signál se uloží do obdobné paměti jaká je na vysílací straně, ovšem nyní je signál do paměti ukládán po sloupcích a vyčítán po řádcích. Na výstupu z paměti je pořadí bitů A1, A2, A3, ... E6, E7, E8, opět zcela shodné s pořadím bitů vstupního signálu.

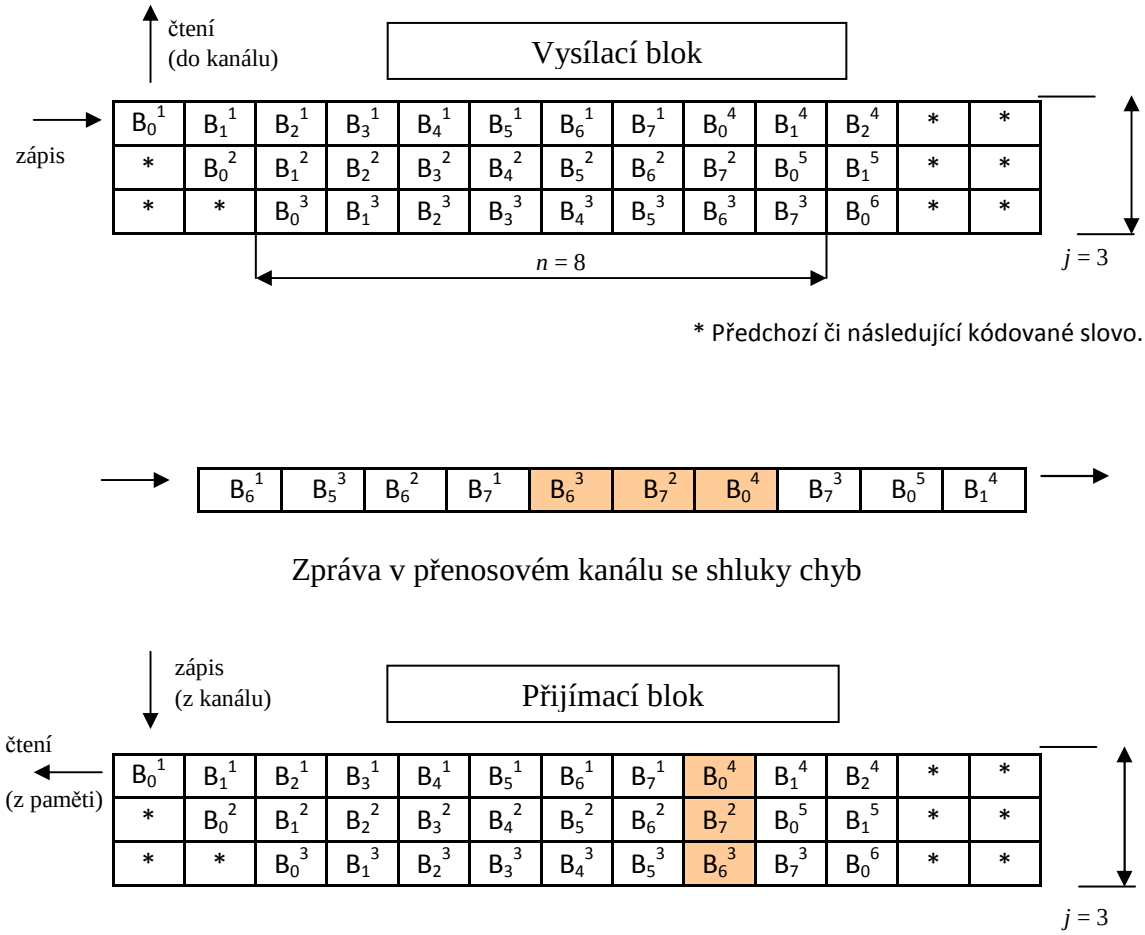
Uskutečněné operace ukládání a vyčítání z pamětí, nemají na užitečný signál žádný vliv. Avšak pokud dojde k rušení signálu a v důsledku vznikne skupinová chyba (v Obr. 2.4 zasáhla skupinová chyba bity A4, B4, C4, D4, E4 – jsou barevně vyznačeny), je v důsledku ukládání a vyčítání signálu z paměti na přijímací straně dosaženo toho, že se skupinová chyba rozprostře. Místo jedné shlukové chyby se vytvoří několik chyb ojedinělých, přičemž počet chybných bitů zůstane stejný. Ovšem signál s ojedinělými chybami lze snáze opravit a již může být aplikován vhodný korekční kód, který předtím neměl šanci provést úspěšnou korekci. Většinou se tento systém používá jako doplněk blokového kódování [2], jenž je účinné proti ojedinělým chybám.

2.2.2 Konvoluční metoda prokládání

Rozdíl mezi konvolučním a bitovým prokládáním je stejný jako rozdíl mezi blokovým a konvolučním kódováním [29]. U bitového prokládání probíhá práce po blocích, které jsou načítány do matice. U konvolučního prokládání celý mechanismus probíhá opět průběžně. Bitový tok, nejčastěji rozdělený do značek, se ukládá do jednotlivých paměťových větví konvolučního prokladače. Po naplnění všech větví n prokladače se vrací na začátek a v tu chvíli dochází k přímému propojení mezi vstupem a výstupem. Tím dochází k různému zpoždění značek a jejich vzájemnému promíchání při "nulovém" zpoždění mezi vstupem a výstupem. Pro zpoždění konvolučního prokládání platí dle [30]:

$$D_{CI} = \frac{j-1}{n} + 2\frac{j-1}{n} + ... + (n-1)\frac{j-1}{n} = \frac{(j-1)*(n-1)}{2}. \qquad (2.6)$$

Konvoluční prokládání multiplexuje výstupy vycházející z kodéru, kde se využívá zpoždění každého následujícího vstupu o délce $\frac{j-1}{n}$, kde j představuje hloubku prokládání a n délku prokládání. Základní princip konvolučního kódování je možné pochopit z Obr. 2.5 - data se v prokladači ukládají po diagonále, pak se vyčítají po sloupcích. Sloupec lze přečíst jakmile jsou všechny jeho buňky zapsány.

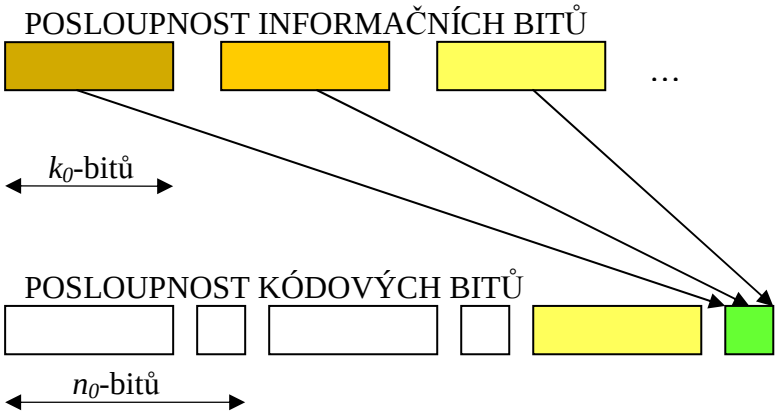


Obr. 2.5: Základní princip konvoluční prokládání.

Mezi důležité parametry představeného zpracování signálu patří rámec vnějšího kódu a hloubka prokládání, jak pro blokovou tak pro konvoluční metodu. Rámec vnějšího kódu udává počet bitů, po kterých se budou opakovat vzniklé ojedinělé chyby. Čím větší bude skupinová chyba, kterou je schopen daný prokládací stupeň rozprostřít, tím větší musí být hloubka prokládání. Interleaving lze využít i při prokládání jednotlivých symbolů [11]. Tyto vnější i vnitřní prokládací stupně často používají složité digitální radiokomunikační systémy, pro lepší ochranu před chybami [29].

2.3 Konvoluční kódy

Konvoluční kodéry lze popisovat jako zdroje zpráv s pamětí. Generace kódových slov probíhá na základě obsahu rámce několika vstupních slov viz. Obr. 2.6. Způsob kódování určité informační posloupnosti tedy závisí nejenom na aktuální vstupní informační posloupnosti, ale též na několika předchozích vstupních slovech [36].

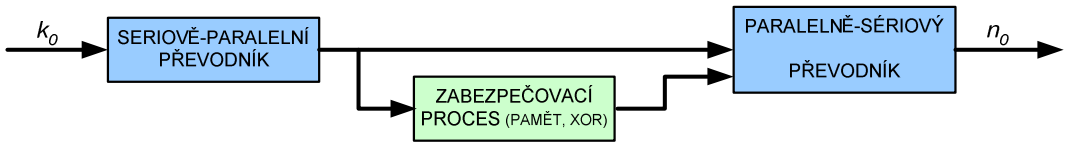


Obr. 2.6: Princip kódování konvolučními kódy.

Při využívání pamětí pro odvození výstupního kódového slova na straně vysílače lze dosáhnout daleko menšího nárůstu potřebného množství redundantních dat na zabezpečení, než je tomu u kódů blokových. Ovšem paměti je třeba využít i na straně přijímače ke kontrole přijatých dat a zde již dochází ke zpožďování užitečného signálu [49]. Konvoluční kód představuje nepřetržitý způsob zabezpečení přenášených posloupností signálových prvků proti chybám.

2.3.1 Systematické konvoluční kódy

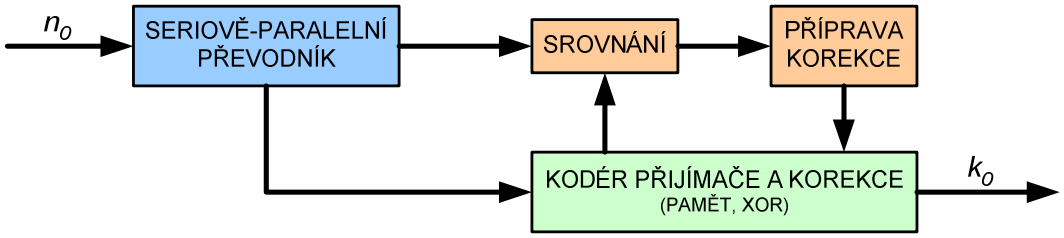
Na vstupu kodéru se rozdělí sériový tok na k_0 vstupních dílčích paralelních toků. V jedné větvi se uskuteční zabezpečující proces pomocí kombinačních obvodů, tj. jsou určeny zabezpečující bity. Na výstupu pak je n_0 dílčích výstupních posloupností, které se opět pomocí slučovače převedou zpět na sériový tok viz. Obr. 2.7, kde je naznačena jedna z variant řešení pomocí tzv. paralelní vstupní paměti. Konvoluční kodéry se nejčastěji realizují posuvnými registry s odlišnou rychlostí posuvu [5]. V jednom kroku, kdy se vytvoří kódové slovo, dojde k posuvu na vstupu o k_0 zdrojových symbolů, ovšem na výstupu bude posuv o n_0 kódových symbolů.



Obr. 2.7: Obecný konvoluční kodér (paralelní zapojení).

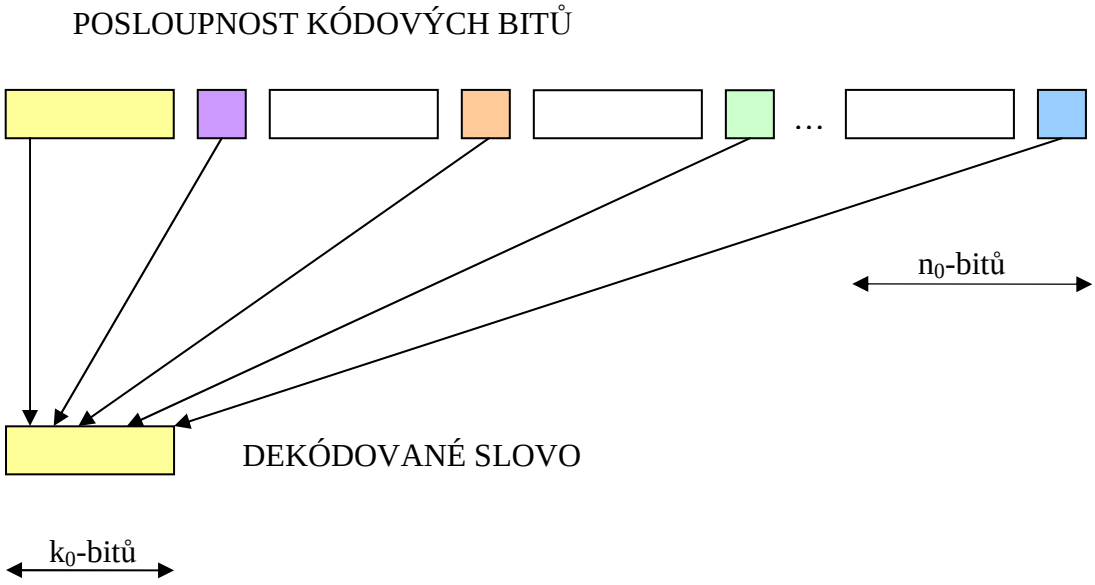
V dekodéru, viz. Obr. 2.8, se vstupní data dělí na informační a zabezpečující část. Informační část představuje vlastní přenášená data, která mohou být poškozena. K ověření korektnosti přijatých dat, se proto opět počítají zabezpečující data. Dojde zde ke srovnání kontrolního bitu ze vstupu dekodéru (bit odvozený z posloupnosti v paměti kodéru) a odpovídajícího vypočítaného bitu z posloupnosti v paměťovém poli dekodéru [44]. Pokud je zjištěn nesoulad vypočítává se za pomocí syndromů poloha chyby, která je následně opravena. Při dekódování se dekódované slovo získává z informačních bitů

právě přenášeného kódového slova a kontrolních bitů celé skupiny, jenž se porovnávají.



Obr. 2.8: Obecný konvoluční dekodér (paralelní zapojení).

Je patrné, že dekódovaná informace bude k dispozici až po dokončení přenosu celé skupiny bitů, ze kterých jsou vytvářeny zabezpečující bity viz. Obr. 2.9. Dekódované slovo se získá až po uplynutí dopravního zpoždění, jenž vzniká zpracováním úseku zprávy a případným použitím sériového přenosu zprávy.



Obr. 2.9: Princip konvolučního dekódování.

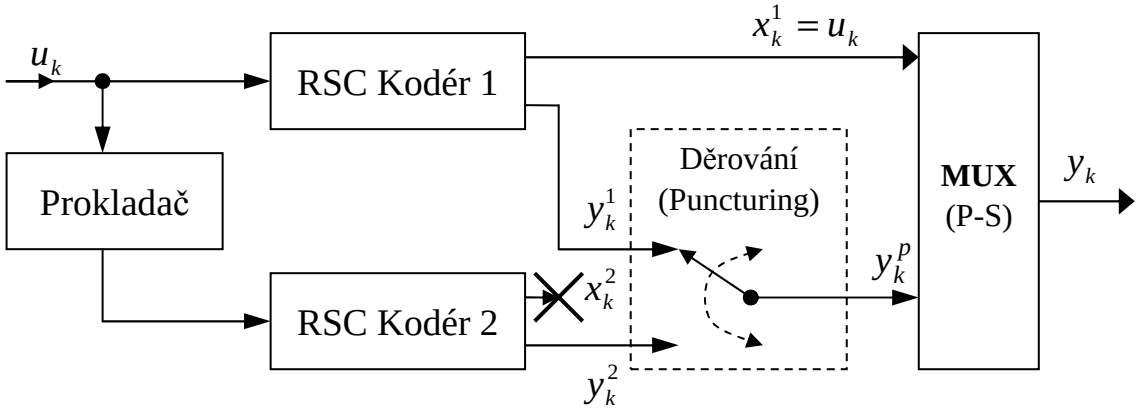
Konvoluční kódy mají schopnost opravit shluky chyb *b*, mezi kterými se nachází určitý ochranný interval *A*, kde informace musí být přenesena bezchybně, což představuje nevýhodu tohoto způsobu zabezpečení. Jinak by nedošlo ke korektní opravě a navíc by se chyba rozšířila i na dříve správně přenesenou část, obdobně jako u kódů blokových [39]. Kód nevyžaduje speciální zabezpečování přenosu kontrolních bitů, neboť stejně detekuje i chybu v kontrolním součtu. Tato skupina kódů není tak známá, jako předchozí představené varianty, avšak v individuálních protichybových systémech může dosáhnout lepších výsledků [16] než první dvě výše uvedené v současnosti hojně používané metody. Na opomíjení konvolučních kódů lze usuzovat

dle výskytu a dostupnosti literatury i technických zpráv pojednávající o dané problematice. Proto této problematice bude podrobněji věnována další část práce.

2.3.2 Turbo kódy

Turbokódy vychází z představy paralelně zřetězených kódů (vzájemně oddělených blokem prokládání) a iterativního způsobu dekódování. Základní myšlenka spočívá ve využití kombinace jednoduchých konvolučních kódů v paralelním zřetězení tak, aby každý z těchto kódů mohl být dekódován odděleně v méně složitém dekodéru. Každý z dekodérů navíc těží ze vzájemné výměny informací mezi jednotlivými dekodéry. Turbokódy umožňují dosáhnout nízké chybovosti BER (Bit Error Rate) i při hodnotách SNR (Signal to Noise Ratio), které leží velmi blízko Shannonovu limitu [9].

Ve většině případů se turbokodér skládá ze dvou (případně z více) paralelně zřetězených konvolučních kódů. V praxi se nejčastěji využívají dva identické rekurzivní systematické konvoluční kodéry (RSC), jenž jsou od sebe vzájemně odděleny prokladačem. Na Obr. 2.10 je ukázáno obecné blokové schéma turbokodéru.



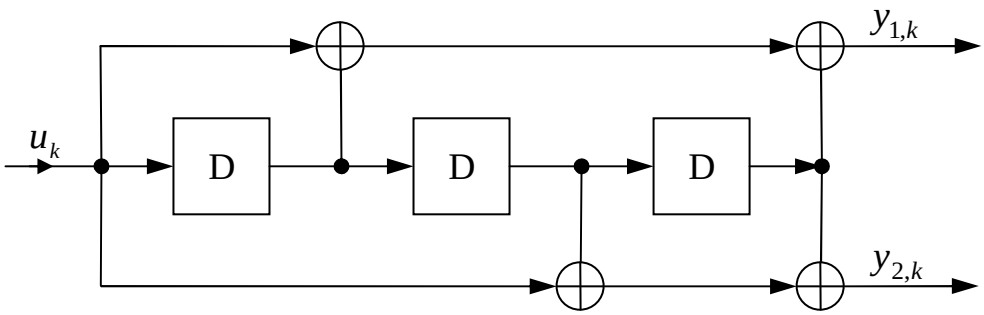
Obr. 2.10: Obecné blokové schéma kodéru turbokódu.

RSC kodér se získá z klasického nerekurzivního konvolučního kodéru (NRC). Použije se jeden z jeho zabezpečovacích výstupů a zavede se zpět na jeho vstup jako zpětnovazební smyčka a druhý výstup se musí nastavit tak, aby poskytoval systematický výstup ($x_k = u_k$).

Základní prvek NRC kodéru představuje posuvný registr složený z několika paměťových buněk, které v sobě uchovávají předchozí hodnoty vstupního bitového toku. Jednotlivé výstupy NRC kodéru jsou tvořeny lineární kombinací současných a předchozích vstupních hodnot. Na Obr. 2.11 lze vidět příklad zapojení NRC kodéru s informační rychlostí $R = 1/2$ a délkou kódového ohraničení $K = 4$. Délka kódového ohraničení udává, jak dlouho se daný jednotlivý bit ze vstupní nezabezpečené posloupnosti podílí na zabezpečovacím procesu a pro jeho výpočet platí vztah:

$$K = m_c + 1, \tag{2.7}$$

kde m_c značí počet paměťových buněk kodéru. Uvedené zapojení je nesystematické, což znamená, že vstupní data nejsou přímo odesílána na některý z paritních výstupů [9]. Obecně však mohou být NRC kodéry konstruovány jako systematické i jako nesystematické.



Obr. 2.11: Zapojení NRC kodéru s R = 1/2 a K = 4.

Proces zabezpečení začíná přivedením vstupní nezabezpečené posloupnosti $u = \{u_1, u_2, u_3, ..., u_N\}$ na vstup turbokodéru. Informační rychlost RSC kodérů dosahuje typicky hodnoty $R = 1/2$. Vstupní data jsou přivedena do prvního RSC kodéru přímo a do druhého RSC kodéru přes prokladač viz. Obr. 2.11. Ten převede přijatou vstupní posloupnost na novou posloupnost, která bude mít pseudonáhodný tvar. Každý z RSC kodérů na svém výstupu poskytuje systematickou a zabezpečovací (paritní) posloupnost. Systematická část z druhého RSC kodéru není dále přenášena, protože ji lze v dekodéru následně obnovit ze systematické části prvního RSC dekodéru.

Pro zlepšení výsledné informační rychlosti turbokodéru, je možné použít blok děrování (puncturing). Děrování slouží k redukci počtu bitů ve výstupním kódovém slově turbokodéru a tím i šířky pásma potřebné k přenosu zakódované informace. Zmíněná operace ovšem představuje nevýhodu ve snížení celkové výkonnosti daného turbokódu, proto je nutné najít určitý kompromis mezi konečnou délkou kódového slova a požadovanou zabezpečovací schopností daného turbokódu.

Blok děrování podle daných pravidel odstraní některé bity z paritních posloupností y_k^1 a y_k^2. Takto vzniklá zabezpečující posloupnost y_k^p se společně se systematickou částí z prvního RSC kodéru přivádí na paralelně-sériový převodník (P-S). Při informační rychlosti obou RSC kodérů $R = 1/2$ a bez použití bloku děrování je výsledná informační rychlost turbokodéru rovna $R_v = 1/3$. Pro každý vstupní datový bit u_k se získá na výstupu turbokodéru trojice bitů $y_k = \{x_k^1, y_k^1, y_k^2\}$. Výstupní zabezpečenou posloupnost lze popsat rovnicí:

$$Y = \{y_1, y_2, y_3, ..., y_N\} = \{x_1^1, y_1^1, y_1^2, x_2^1, y_2^1, y_2^2, ..., x_N^1, y_N^1, y_N^2\}, \qquad (2.8)$$

kde N je délka vstupní nezabezpečené posloupnosti u.

Pokud se blok děrování využije a je nastaven například tak, aby vybíral jenom liché paritní bity z prvního RSC kodéru a sudé paritní bity z druhého RSC kodéru, výsledná informační rychlost je rovna hodnotě $R_v = 1/2$ a v čase k bude výstup turbokodéru $y_k = \left(x_k^1, y_k^p\right)$. Celkový výstup z turbokodéru popisuje v tomto případě rovnice:

$$Y = \{y_1, y_2, y_3, ..., y_N\} = \left\{x_1^1, y_1^1, x_2^1, y_2^2, x_3^1, y_3^1, ..., x_N^1, y_N^z\right\}, \tag{2.9}$$

přičemž $z = 1$, pokud N bude liché číslo a naopak $z = 2$, pokud N bude číslo sudé. Dekódování lze provádět pomocí Viterbiho algoritmu, SOVA algoritmu (soft-output Viterbi algoritmus), MAP algoritmu (Maximum a-posteriori) [32] a [35].

Návrh vhodného prokladače představuje klíčový faktor, který ovlivňuje celkovou výkonnost turbokódů. Hlavním úkolem prokladačů v tomto případě je zajistit, společně s RSC kodéry, aby Hammingova váha výsledného kódové slova byla velká i v případech, kdy se na vstupu turbokodéru objeví některá z "nejméně vhodných" datových sekvencí. Složitost i náročnost uváděného kódování je opět velmi vysoká a celková informační rychlost příliš nízká. Vhodněji lze pro některé případy použít systémy opravující menší shluky chyb, za to však s rychlejším kódováním přenášených dat a docílit vyšší informační rychlosti.

3 Protichybové kódování

Chybovostí v systému se zabýváme, až pokud výskyt různých typů chyb překročí únosnou mez. Jelikož zavedení protichybového kódování představuje zvýšení složitosti celého systému a snížení dosahované přenosové informační rychlosti. Ovšem z důvodu šumu v komunikačních kanálech se jedná o vhodný způsob potlačení nežádoucích vlivů. Shlukové chyby jsou v oblasti miniaturizace místa pro záznam dat a zrychlování přenosu dat problémem, který mění zásadním způsobem pohled na zabezpečení užitečné informace před nežádoucími účinky šumu [31]. Ukazuje se, že pro tuto skupinu chyb, již nejsou příliš vhodné systémy pro opravu náhodných chyb, které se momentálně hojně využívají [20].

Vzhledem k aktuálnímu vývoji v oblasti komunikačních a počítačových technologií a srovnání výhod i nevýhod jednotlivých technik popsaných v kapitole 2 se nabízí možnost zaměřit se na vhodnou volbu konvolučního kódu pro kompenzaci vlivů šumu u individuálních protichybových systémů. Existuje určitá skupina konvolučních kódů, které jsou vhodněji uzpůsobeny k opravě shlukových chyb, než obvyklé systémy pro opravu vícenásobných chyb. Především mají menší nárůst potřebného množství redundantních dat na zabezpečení i dosahují vyšší informační rychlosti, než je tomu u klasických kódů blokových.

Na druhou stranu u sofistikovaných systémů založených na Turbo-kódech či Reed-Solomonových kódech, jenž slouží k opravě velmi dlouhých shluků chyb dochází při menších shlucích chyb k nevyužitelnosti jejich robustnosti. Jelikož opravy dlouhých shluků chyb souvisí s komplikovanými výpočetními algoritmy jak pro kódování tak především pro dekódování [12]. Tedy i zde je jistá oblast, kde využitelnost zmíněných systémů není ideální a vhodnější volbu opět představují jednodušší konvoluční kódy pro opravu shluků chyb.

Zabezpečení se dosahuje zvýšením nadbytečnosti na úkor vlastního přenosu užitečné informace. Složitost a náročnost realizace zabezpečení představuje taktéž omezující faktor použitelnosti. V jednotlivých případech je tedy velmi důležité najít nejvhodnější řešení, které se ovšem liší případ od případu. Avšak není doposud vypracována metodika pro posouzení optimálního kódového zabezpečení v jednotlivých individuálních případech.

V současnosti je problém shlukových chyb převážně řešen použitím blokových kódů s prokládáním či RS kódů. Záměrem je vytvoření rozšířené metodiky pro návrh individuálních protichybových systémů využívajících alternativního řešení korekce shlukových chyb pomocí vybraných konvolučních kódů. Rozšíření popisu systematických konvolučních kódů pomocí matematického aparátu umožní dokonalejší posuzování jednotlivých kódových zabezpečení i srovnání s dosud hromadně používanými univerzálními řešeními.

Při návrhu v individuálních protichybových systémů musí konstrukce kódovacího a dekódovacího zařízení sledovat několik základních cílů:

1. Rychlé kódování informace.

2. Snadný přenos zakódované zprávy.

3. Rychlé dekódování přijaté zprávy.

4. Opravu chyb způsobených šumem v kanálu během přenosu zprávy.

5. Maximalizaci množství informace přenesené za jednotku času.

Hlavním bodem je čtvrtý z těchto úkolů. Problém spočívá v tom, že dosažení čtvrtého cíle není v souladu s pátým cílem, a nemusí být ani příliš v souladu s prvními třemi uvedenými úkoly. Jakékoliv řešení tohoto problému je nutně kompromisem mezi těmito pěti cíly. Na základě zjištěných výsledků lze hledat variantní řešení pro již používané způsoby protichybových zabezpečení s ohledem na dosažení vyšší efektivity výsledků.

3.1 Řešení u individuálních protichybových systémů

Stále rostoucí integrace služeb vyžaduje od jednotlivých komunikačních koncových zařízení podstatnou univerzálnost. Existující koncová zařízení musí být schopna využívat různorodé technologie přístupových sítí. Možnosti realizace připojení koncových zařízení jsou v současnosti velmi rozmanité. Tím je nutně dána i různá odolnost proti poškození informačního obsahu přenášených zpráv [25].

Rozvoj technologií a číslicové techniky zaznamenaný v posledních desetiletích umožňuje implementaci stále náročnějších algoritmů. Otevírají se možnosti provedení postupů popsaných teoreticky v minulosti, které tehdy nebylo možné kvůli nedostatku výpočetní kapacity číslicových obvodů prakticky realizovat. Jednou z oblastí, jenž využívá stále zlepšujících se vlastností integrovaných obvodů, je digitální zpracování signálu, které hraje důležitou roli nejen při v komunikacích, ale i úschově a ochraně informace. S rostoucími nároky na přenosové rychlosti moderních bezdrátových technologií, které většinou využívají pro přenos rádiových frekvencí (jako např. WiFi, WiMAX), roste i náročnost zpracování číslicového signálu [20].

Vývoj kráčí dopředu především směrem speciálních robustních systémů se složitými výpočetními algoritmy a snaží se tak pokrýt většinu požadavků [4]. Avšak u individuálních protichybových systémů, kde důraz je kladen i na malou výpočetní náročnost, ať již kvůli ceně zařízení či kvůli většímu množství přenesených dat, tyto systémy nejsou vždy tím nejvhodnějším možným řešením. Jelikož jednodušší systémy mohou poskytnout v některých případech mnohem vyšší požadovanou efektivitu [16].

Problémem shluků chyb se v současných systémech musí zabývat stále více zařízení. Proto v individuálních protichybových systémech je vhodné zvážit použití jiných zabezpečovacích kódových technik, než masová aplikace stávajících řešení pro

dosažení maximální efektivity. I z toho důvodu, že současné komunikační kanály a informační systémy kladou stále větší nároky na kvalitu zabezpečení přenášených informací.

U individuálních protichybových systémů se vyhledávají specifická řešení, jenž dokáží poskytnout lepší vlastnosti než univerzální varianty. Teorie zabezpečovacího kódování představuje rozsáhlý soubor poznatků. Avšak v dosud dosažitelné literatuře se lze setkat především z popisem jednotlivých hojně užívaných kódů, zatímco jejich vhodný výběr, srovnání různých variant řešení a popřípadě způsoby realizace vlastního zabezpečení zůstávají stranou.

U problematiky specifických řešení se analýza shlukových chyb opírá o možnosti modelování a následné simulace dějů kódování, dekódování a přenosu v síti za pomocí vhodných softwarových nástrojů [14]. Význam získaných výsledků přitom závisí kromě vhodné volby simulačního prostředí také na podrobnosti i dostatečné věrnosti modelů [27], jenž by měly napomoci při návrhu realizace daného protichybového systému a především slouží k ověření navrženého teoretického zapojení. Hlavní zaměření disertační práce se zabývá zmíněnými problémy v této kapitole – rozšíření možnosti vyhledávání specifických řešení pro opravu shluků chyb pomocí systematických konvolučních kódů u individuálních protichybových systémů.

4 Aplikace systematických konvolučních kódů

Konvoluční kódy umožňují omezit vliv šumu, jenž způsobuje i vznik shlukových chyb. Na rozdíl od kódů blokových dochází k daleko menšímu rozrůstání potřebného množství zabezpečujících dat [39] a tím je k dispozici větší přenosová informační rychlost. Využití interleavingu je vázáno na předchozí protichybové kódování a tvoří pouze další přídavný systém, jenž navíc způsobuje zpožďování přenášené informace. Samostatné použití zmíněného systému nepřináší žádné zlepšení. Robustní systémy pro opravu shluků chyb představují univerzálnější řešení, mnohdy však jednoduchost systémů využívajících konvolučních kódů umožní efektivnější využití výpočetních prostředků a tím rychlejší přenos dat.

Některé konvoluční kódy jsou vhodněji uzpůsobeny k opravě shlukových chyb [30], jelikož k zabezpečení je využíváno více informačních bloků a není třeba vytvářet takové množství zabezpečujících bitů pro jeden blok. Ovšem je třeba počítat se vznikem zpoždění, jenž souvisí s počtem bloků podílejících se na příslušném zabezpečení. Avšak v některých situacích lze použitím konvolučního kódu dosáhnout lepších výsledků než spojením systému využívající blokový kód s následným prokládáním či použití složitých zřetězených systémů.

Pro vytváření konvolučních kódů se využívají dvě základní metody [6]. Kód lze zadat buď pomocí skupiny vytvářecích mnohočlenů nebo pomocí vytvářecí matice. Při zadávání pomocí vytvářecího mnohočlenu se používá operátor zpoždění, který slouží k vyjádření průchodu signálového prvku posuvným registrem kodéru. Obecnější je zápis pomocí vytvářecí matice, která je definována následovně:

$$\mathbf{F} = \mathbf{P} * \mathbf{G}\,, \tag{4.1}$$

kde řádky matice **P** představují dílčí vstupní toky, **F** dílčí výstupní toky a **G** je vytvářecí matice konvolučního kódu a je polonekonečná. Zadávat kód pomocí vytvářecí matice lze vždy, kdežto vytvářecí mnohočleny představují jen zjednodušený zápis vytvářecí matice a jejich použití je omezené, využívá se především pro cyklické kódy.

Kontrola správnosti přenesených signálových prvků daného konvolučního kódování se provede vynásobením příchozí zprávy **F*** s kontrolní maticí **H**, kterou lze odvodit z vytvářecí matice **G**. Dva nejznámější dekódovací způsoby se rozdělují podle způsobů, jenž využívají [24], na:

- Prahové – pokud určitý počet výsledků kontrolních operací překročí jistý práh, považuje se většinový výsledek za správný. Pro hledání správné posloupnosti je využit generátor syndromu.
- Pravděpodobnostní – je založeno na porovnání přijatého úseku zprávy s úseky ze seznamu užívaných zpráv. Vybere se ten úsek zprávy, který se od přijatého úseku nejméně liší a je tak nejpravděpodobnější. Důležitým parametrem je zpracovávaný úsek.

Nejznámější variantu dekódování konvolučních kódů představuje Viterbiho algoritmus, který po úsecích zpracovává přijatá data a snaží se o nalezení nejpravděpodobnějšího odhadu správné posloupnosti. Princip algoritmu spočívá ve vyhledávání nejmenší vzdálenosti dekódované posloupnosti od přijaté posloupnosti. (U blokových kódů se jedná o Hammingovu vzdálenost.) Nevýhodou zmiňovaného algoritmu je jeho značná náročnost na počet numerických operací [18]. Uvedený způsob se používá jen pro dekódování kódů s relativně krátkými omezujícími délkami [35], jelikož závislost počtu prováděných operací má exponenciální charakter.

Viterbiho algoritmus používá tvrdého rozhodování a hledá nejpravděpodobnější odhad vyslané posloupnosti symbolů. Lepších výsledků, tedy nižší bitové chybovosti, lze dosáhnout pomocí algoritmu MAP, který je ovšem mnohem složitější než Viterbiho algoritmus [28].

Při tvrdém rozhodování (hard decision) je výstup z kanálu kvantován v případě dvoustavového signálu na dvě úrovně, tj. dostáváme pouze úrovně 0 nebo 1. Kvantováním na dvě úrovně při tvrdém rozhodování dochází ke zvýšení úrovně šumu o kvantovací šum a zvyšuje se tím pravděpodobnost chyby oproti měkkému rozhodování. Při měkkém rozhodování (soft decision) se provádí optimalizace. Výstup z kanálu je vzorkován a poté následuje hledání nejpravděpodobnějšího signálového bodu. Zpracování informací získaných při měkkém rozhodování je náročnější než u tvrdého rozhodování, ale lze dosáhnout nižší bitové chybovosti než pro tvrdé rozhodování.

Vhodnější metoda k vyhledávání chyb (i s ohledem na výpočetní náročnost) spočívá v lokalizaci pomocí syndromu při použití principu prahového dekódování [16]. Základním vlastností je nalezení dostatečného počtu kontrolních operací mezi jednotlivými prvky úseku zprávy, které mají mezi sebou tzv. vztah ortogonality vzhledem k prvku, jehož správnost se po přenosu kontroluje. Počet potřebných rovnic je určen požadavkem existence alespoň dvou ortogonálních součtů k jednomu opravovanému bitu. V obou těchto rovnicích se opravovaný bit vyskytuje, zatímco ostatní bity se nevyskytují (nesmí vyskytnout) v obou rovnicích [21].

4.1 Úvod k jednotlivým kódům

Poprvé byly představeny kódy pro opravu shlukových chyb Hagelbargerem. Nezávisle na sobě Iwadari a Massey zkonstruovaly efektivnější kódy stejného typu. Konvoluční kódy pro opravu postupných shlukových chyb byly studovány Wynerem a Ashem. Optimální kódy pro opravu postupných shlukových chyb byly později objeveny nezávisle Berlekampem a Preparatem.

Pro výše uvedené systematické konvoluční kódy bude ukázán návrh kodéru a dekodéru, který je schopen opravit shluk 4 chyb. Při této velikosti shluku chyb jsou kompletní schémata zapojení stále ještě dostatečně přehledná. Díky systematičnosti ve výsledné přenášené kódové kombinaci lze vždy rozlišit prvky informační a zabezpečující. Jednotlivé kódy jsou popsány především vytvářecí blokovou maticí $\mathbf{B_0}$.

Bloková matice je specifická pro každý kód a lze pomocí ní odvodit jednotlivé prvky kodéru i dekodéru a jejich následné potřebné zapojení bez ohledu na velikost opravovaných shluků chyb. Po odvození rozsáhlého matematického aparátu, jenž podrobněji charakterizuje dané kódy, lze hledat nejlepší variantu mezi touto skupinou kódů. Při využití získaných vztahů pro celkové zpoždění způsobené průchodem zprávy kodekem *Z*, celkové konstrukční složitosti kodeku S_L a S_P se zvýší počet srovnávacích kritérií. Vyhodnocením těchto dalších vlastností se dosáhne detailnějšího rozboru než jen při vyhodnocování požadovaného bezchybného intervalu *A*. Jelikož základ srovnání vychází u všech kódů ze stejného informačního poměru *R* a maximální opravitelné délky shluku chyb *b*.

Informační rychlost *R* je definována pomocí krátkých bitových úseků n_0, k_0 a představuje jeden ze základních parametrů konvolučních kódů:

$$R = \frac{k_0}{n_0}, \tag{4.2}$$

kde k_0 představuje vstupní dílčí tok, n_0 odpovídající výstupní dílčí tok. Obdobně je tento parametr definován u kódů blokových a udává, jak se v důsledku použití kódu sníží rychlost přenosu bitů nesoucích původní nezabezpečenou informaci.

Pro snadnější konstrukci následujících kodeků bude využito i skutečnosti, že se vždy jedná o systematické kódy. Rozdělení zprávy na informační a zabezpečující část, jenž následuje až po informačních bitech příslušného přenosu, je provedeno kvůli snadnějšímu technickému řešení kodéru i dekodéru.

4.2 Základní Hagelbargerův kód (n_0; n_0 -1)

Kód lze určit pomocí blokové matice v dekadickém tvaru $\mathbf{B_D}$, která má tvar čtvercové matice rozměru n_0 x n_0 [45]. V každém řádku v místě odpovídající prvku diagonály je na úhlopříčce liché dekadické číslo. Řádky se číslují zespod a v prvním řádku je trojka, v druhém řádku je pětka až v (n_0 - 1) řádku je dekadické číslo ($2n_0$ - 1). Poslední řádek obsahuje jedničku. Pro n_0 = 4 platí vztah:

$$\mathbf{B_D} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 7 & 0 \\ 0 & 5 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{bmatrix} \rightarrow \mathbf{B_0} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{4.3}$$

Matice $\mathbf{B_D}$ se pro potřebu binárních kódů převádí na blokovou matici ve dvojkovém tvaru $\mathbf{B_0}$. Dvojková bloková matice $\mathbf{B_0}$ má opět n_0 sloupců. Počet řádků je určen součinem n_0 a počtem míst potřebných k zapsání největšího čísla, které se v $\mathbf{B_D}$ vyskytuje. Počet míst dekadického čísla n, potřebných pro jeho zapsání dvojkovým číslem, se značí $L(n)$, což je nejmenší celé číslo, které vyhovuje nerovnosti:

$$L(n) \geq 1 + \log_2 n\,. \tag{4.4}$$

Binární čísla s menším počtem míst mají v matici $\mathbf{B_0}$ na nepoužitých místech nuly. První číslo odspodu, jenž je v dekadické podobě blokové matice rovno číslu tři, má vždy pouze dva řádky, protože nuly v místech nepoužitých dvojkových míst by způsobovaly v realizaci kodéru pouze neužitečné časové zpoždění. Pro počet řádků dvojkové matice jde tedy odvodit následující vztah:

$$(n_0 - 1) * L(n) + 2\,, \tag{4.5}$$

a bloková matice $\mathbf{B_0}$ má rozměry:

$$\mathbf{B_0}\left[n_0; n_0 - 1\right] = \left((n_0 - 1) * L(n) + 2; n_0\right). \tag{4.6}$$

Tento kód opravuje shluky chyb délky b bitů:

$$b \leq n_0\,. \tag{4.7}$$

Mezi těmito shluky chyb musí být v bitovém toku určitý bezchybný úsek – ochranný interval A:

$$A \geq n_0^2 * L(n) - 1. \tag{4.8}$$

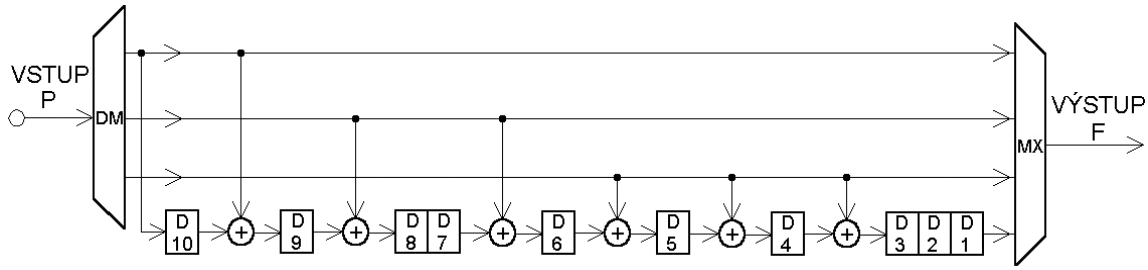
Kód pro opravu shluku chyb $b \leq 4$ bity potřebuje dle (4.8) dodržení délky ochranného intervalu $A \geq 47$ bitů. Vytvoření zapojení kodéru se odvozuje z $\mathbf{B_0}$ pomocí

souboru vytvářecích mnohočlenů. Kódovací obvod je vytvořen tolika paměťovými buňkami, kolik řádků má $\mathbf{B_0}$. Mezi tyto buňky jsou zapojeny sčítačky mod2, dané vytvářecími mnohočleny. U vytvářecího mnohočlenu je výstupním tokem čtvrtý sloupec blokové matice (4.3). Vstupním tokem je postupně první až třetí sloupec této matice. Hledají se řádky, kde příslušný sloupec obsahuje jedničky viz. rovnice (4.9), (4.10), (4.11). Zapojení kodéru jest uvedeno na Obr. 4.1 a určují jej zabezpečující vytvářecí mnohočleny získané z vytvářecí blokové matice $\mathbf{B_0}$:

$$g_1^{(4)} = D^9 + D^{10}, \tag{4.9}$$

$$g_2^{(4)} = D^6 + D^8, \tag{4.10}$$

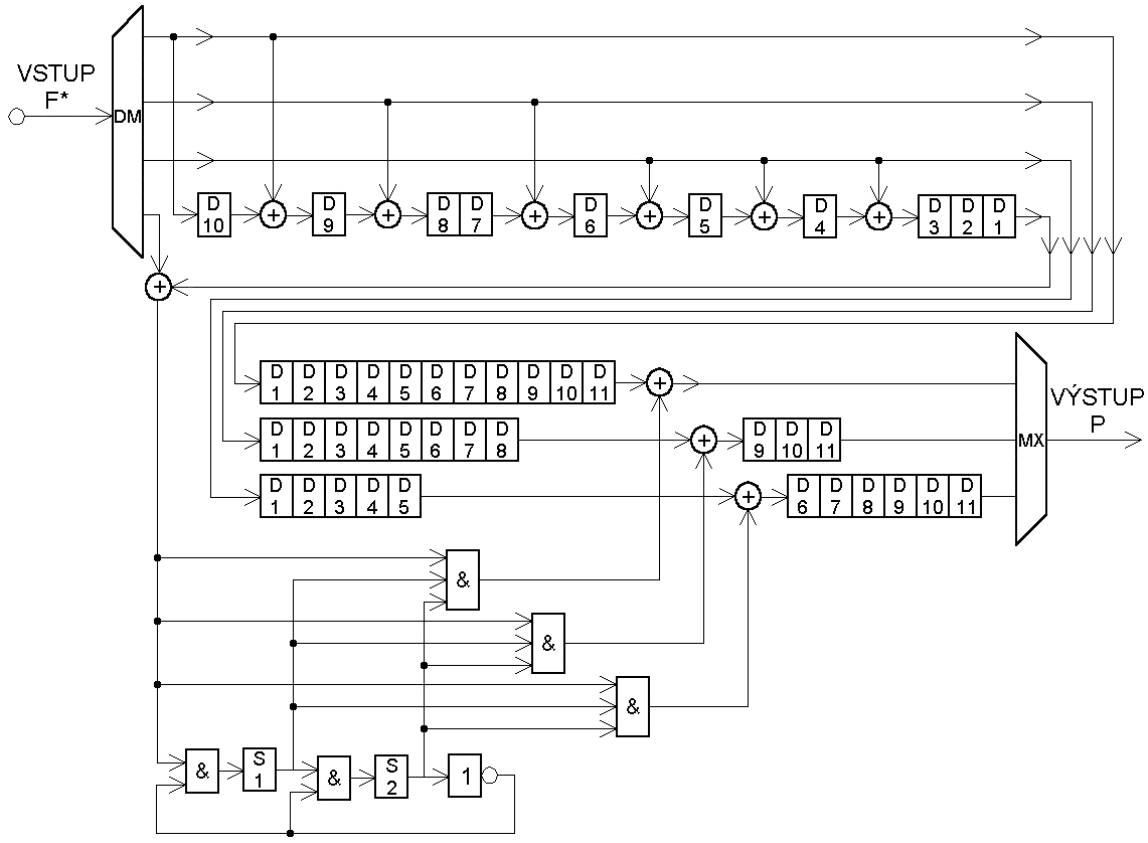
$$g_3^{(4)} = D^3 + D^4 + D^5. \tag{4.11}$$



Obr. 4.1: Kodér - Hagelbargerova kódu pro korekci 4 chyb.

Dekodér Hagelbargerova kódu využívá přímého převodu vektoru syndromu na chybový vektor (viz. Obr. 4.2). Z přeneseného bloku dat se oddělí zabezpečovací prvek, který se sečte mod2 s nově vypočítaným zabezpečovacím prvkem, jenž se vytvořil v kodéru přijímače. Tím vznikne syndromový prvek *s*, který slouží k nalezení a korekci chyb v dílčích posloupnostech přenesených informačních prvků.

S využitím získaných hodnot i na základě konkrétních schémat zapojení lze odvodit další parametry, jenž lépe rozšíří popis daného systému [46]. Účelem je získat co nejobecnější vztahy, které budou mít všeobecnou platnost. Jejich využití jistě nalezne uplatnění při návrzích individuálních protichybových systémů, kdy bez předchozí podrobné znalosti daného systému se získá více popisných informací. Může tak dojít při budoucích návrzích k úspoře času vyřazením nevhodných variant na základě zjištěných hodnot.



Obr. 4.2: Dekodér Hagelbargerova kódu pro korekci 4 chyb.

Základní zpoždění je způsobováno především průchodem bitů přes paměťové buňky. V kodéru dochází k průběžnému tvoření zabezpečujících bitů a použité paměťové buňky slouží jen k tomuto účelu. Přenášená informace tak zde není vůbec zpožďována. V dekodéru se avšak některé paměťové buňky používají k úschově přenesené informace za účelem případné korekce. Vlastní užitečná informace není ze vstupu dekodéru ihned k dispozici na výstupu dekodéru a již dochází ke zpoždění. Pro celkové základní zpoždění Hagelbargerova kodeku při paralelním přenosu platí vztah:

$$Z = Z_K + Z_D = Z_D = L(n) * n_0 - 1, \tag{4.12}$$

kde Z_K je zpoždění kodéru a Z_D zpoždění dekodéru. V případě sériového přenosu se v dekodéru uplatní jednotlivé větve, mezi které demultiplexor přijatý informační signál rozděluje:

$$Z_S = Z * (n_0 - 1) = L(n) * (n_0^2 - n_0) - n_0 + 1. \tag{4.13}$$

Z Obr. 4.1 a Obr. 4.2 je zřejmé, že pro realizaci zapojení se používají především dva typy prvků a to v různém množství, i když kodér má evidentně složitější strukturu. Konstrukční složitost kodeku je dána především potřebným počtem paměťových buněk a logických operátorů. Multiplexory-demultiplexory na vstupu-výstupu systému netřeba

uvažovat, jelikož ty zajišťují úpravu rychlosti zpracování, s čímž musí pracovat každý zabezpečovací kód.

Pro počet paměťových buněk kodéru S_{PK} platí:

$$S_{PK} = L(n) * (n_0 - 1) + 1. \tag{4.14}$$

Při stanovení počtu paměťových buněk dekodéru je situace složitější. Systém dekodéru lze rozdělit na jednotlivé dílčí oblasti – kodér přijímače S_{PD1}, korekce S_{PD2} a příprava korekce S_{PD3}:

$$S_{PD1} = S_{PK} = L(n) * (n_0 - 1) + 1, \tag{4.15}$$

$$S_{PD2} = (L(n) * n_0 - 1) * (n_0 - 1), \tag{4.16}$$

$$S_{PD3} = L(n) - 1. \tag{4.17}$$

Součtem se pak dostane celkový počet paměťových buněk dekodéru:

$$S_{PD} = S_{PD1} + S_{PD2} + S_{PD3} = L(n) * n_0^2 - n_0 + 1, \tag{4.18}$$

a pro počet paměťových buněk celého kodeku platí:

$$S_P = S_{PK} + S_{PD} = \left(L(n) * (n_0^2 + n_0 - 1)\right) - n_0 + 2. \tag{4.19}$$

Druhou skupinu tvoří jednotlivé logické operátory, jenž se podílí na zajišťování vlastní ochrany přenášené informace. Pro počet logických operátorů kodéru S_{LK} platí:

$$S_{LK} = 2(n_0 - 1). \tag{4.20}$$

Celý dekodér je vhodné opět rozdělit na jednotlivé dílčí části – kodér přijímače S_{LD1}, korekce S_{LD2} a příprava korekce S_{LD3}:

$$S_{LD1} = S_{LK} = 2(n_0 - 1), \tag{4.21}$$

$$S_{LD2} = n_0 - 1, \tag{4.22}$$

$$S_{LD3} = 2(n_0 - 1) + 1. \tag{4.23}$$

Součet částí udává počet logických operátorů dekodéru:

$$S_{LD} = S_{LD1} + S_{LD2} + S_{LD3} = 5(n_0 - 1) + 1, \tag{4.24}$$

celý kodek pak obsahuje:

$$S_L = S_{LK} + S_{LD} = 7(n_0 - 1) + 1. \tag{4.25}$$

4.3 Základní Iwadari-Masseyho kód ($m \cdot n_0$; $m \cdot k_0$)

Iwadari-Masseyho kód je popsán svojí vytvářecí blokovou maticí $\mathbf{B_0}$ [47], která má n_0 sloupců. Sloupce jsou indexovány zleva od a_i, řádky se číslují vzestupně zespodu. Na nejvyšším řádku úplně vpravo je umístěna jednička představující zabezpečovací prvek. Následují nulové řádky až do počtu n_0. V dalším řádku je umístěna jednička posunutá oproti původní pozici o jedna doleva. Následující řádek má jedničku na stejné pozici. Takto je zajištěna oprava jednoho bitu. Pro opravu dalších bitů je třeba se opět posunout o jednu pozici vlevo a počet vložených nulových řádků mezi dvěma jedničkami ve stejném sloupci vždy vzrůstá o jeden řádek až do celkové velikosti n_0. Názorně vše uvádí vztah (4.26), kde je obecné schéma vytvářecí blokové matice $\mathbf{B_0}$ Iwadari-Masseyho kódu:

$$\mathbf{B_0} = \begin{array}{|ccccccc|l} 0 & 0 & \ldots & 0 & 0 & 1 & \\ 0 & 0 & \ldots & 0 & 0 & 0 & \\ 0 & 0 & \ldots & 0 & 0 & 0 & \\ \multicolumn{7}{|c|}{\vdots} & n_0 \text{ řádků} \\ 0 & 0 & \ldots & 0 & 0 & 0 & \\ 0 & 0 & \ldots & 0 & 1 & 0 & 2 \text{ řádky} \\ 0 & 0 & \ldots & 0 & 1 & 0 & \\ 0 & 0 & \ldots & 1 & 0 & 0 & 3 \text{ řádky} \\ 0 & 0 & \ldots & 0 & 0 & 0 & \\ 0 & 0 & \ldots & 1 & 0 & 0 & \\ \multicolumn{7}{|c|}{\vdots} & n_0\text{-}1 \text{ řádků} \\ 0 & 1 & \ldots & 0 & 0 & 0 & \\ 1 & 0 & \ldots & 0 & 0 & 0 & \\ 0 & 0 & \ldots & 0 & 0 & 0 & \\ \multicolumn{7}{|c|}{\vdots} & n_0 \text{ řádků} \\ 0 & 0 & \ldots & 0 & 0 & 0 & \\ 1 & 0 & \ldots & 0 & 0 & 0 & \\ \hline a_i & b_i & & x_i & y_i & z_i & \end{array} \quad . \tag{4.26}$$

Bloková matice $\mathbf{B_0}$ má rozměry:

$$\mathbf{B_0}\left[m \cdot n_0 ; m \cdot k_0\right] = (m; n_0), \tag{4.27}$$

kde jednotlivé parametry kódu m a k_0 jsou definovány:

$$m = \frac{n_0 * (n_0 - 1)}{2} + (2n_0 - 1), \tag{4.28}$$

$$k_0 = n_0 - 1, \tag{4.29}$$

k_0 a n_0 představují počet dílčích vstupních či výstupních paralelních toků, m udává počet řádků příslušné vytvářecí matice. Při návrhu jsou třeba ještě parametry pro korekční schopnost b a ochranný interval A:

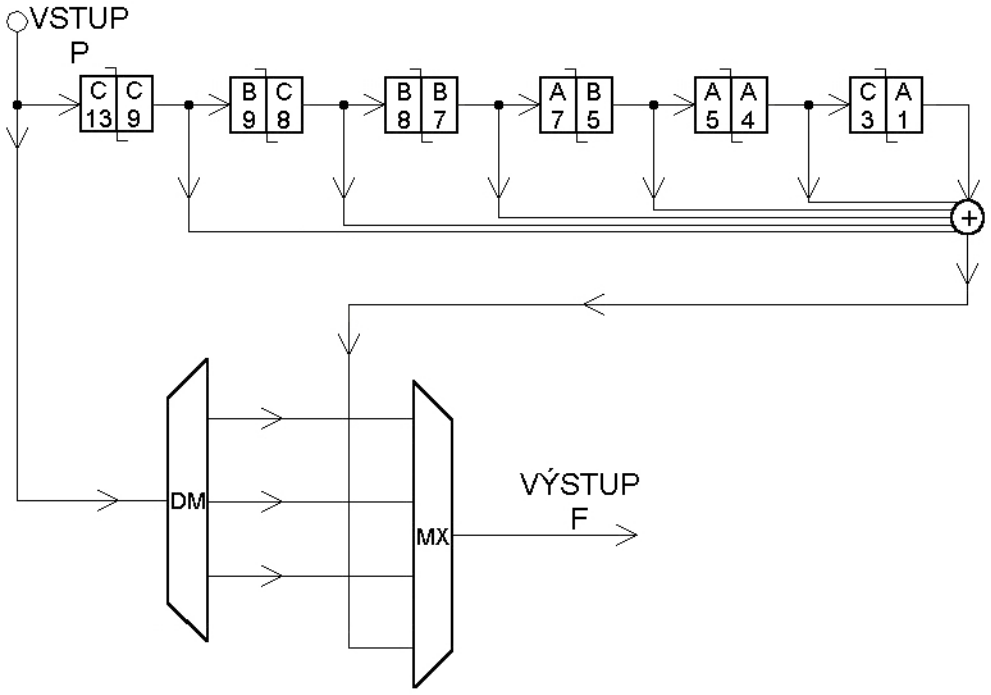
$$b \le n_0, \tag{4.30}$$

$$A \ge n_0 * m - 1. \tag{4.31}$$

V případě překročení mezní zabezpečovací schopnosti nezpůsobuje tento kód nekonečný průnik chyby do vlastní informace [50]. Z vytvářecí blokové matice lze sestavit celý kodek Iwadariho kódu viz. Obr. 4.3 a Obr. 4.4, kde vyobrazená schémata opět zabezpečují zprávu před shlukem chyb velikosti $b \leq 4$. Základem je bloková vytvářecí matice $\mathbf{B_0}$ určená ze vztahu (4.26). Pro vytvoření návrhu zapojení je důležitá syndromová rovnice matice kódu, která udává jednotlivé prvky kodéru. Za předpokladu, že v samotném kodéru nevznikají chyby, pak ze vztahu (4.26) pro zabezpečovací prvek d_{13} platí:

$$d_{13} = a_1 + a_4 + b_5 + b_7 + c_8 + c_9. \tag{4.32}$$

Získaná rovnice (4.32) přímo určuje schéma zapojení kodéru viz Obr. 4.3. V obrázku kodéru i dekodéru je pro názornost vyznačená jen první a poslední paměťová buňka daného úseku. Tedy úsek C 13 až C 9 obsahuje všechny následující paměťové buňky: C 13, B 13, A 13, C12, …, B 10, A 10, C 9.



Obr. 4.3: Kodér Iwadari-Masseyho kódu pro korekci 4 chyb.

Způsob dekódování dat na straně příjemce je naznačen na Obr. 4.4. Princip dekodéru spočívá ve využití prahového dekódování. K syndromové rovnici pro určitý čas k bitům v ní obsažených je třeba nalézt bity na zabezpečení se podílejících v čase předcházejícím. Za pomocí dvou syndromových rovnic lze provést korekci jednoho určitého bitu. Pro korekci dalších bitů je třeba jít dále hlouběji do historie a určit tak další nezbytné syndromové rovnice. Jejich celkový počet určuje množství potřebných

syndromových paměťových buněk. Model dekodéru (viz. Obr. 4.4) je sestaven na základě znalostí potřebných syndromových rovnic:

$$s_{13} = a_1 + a_4 + b_5 + b_7 + c_8 + c_9 + d_{13}, \tag{4.33}$$

$$s_{12} = a_0 + a_3 + b_4 + b_6 + c_7 + c_8 + d_{12}, \tag{4.34}$$

$$s_{11} = a_{-1} + a_2 + b_3 + b_5 + c_6 + c_7 + d_{11}, \tag{4.35}$$

$$s_{10} = a_{-2} + a_1 + b_2 + b_4 + c_5 + c_5 + d_{10}, \tag{4.36}$$

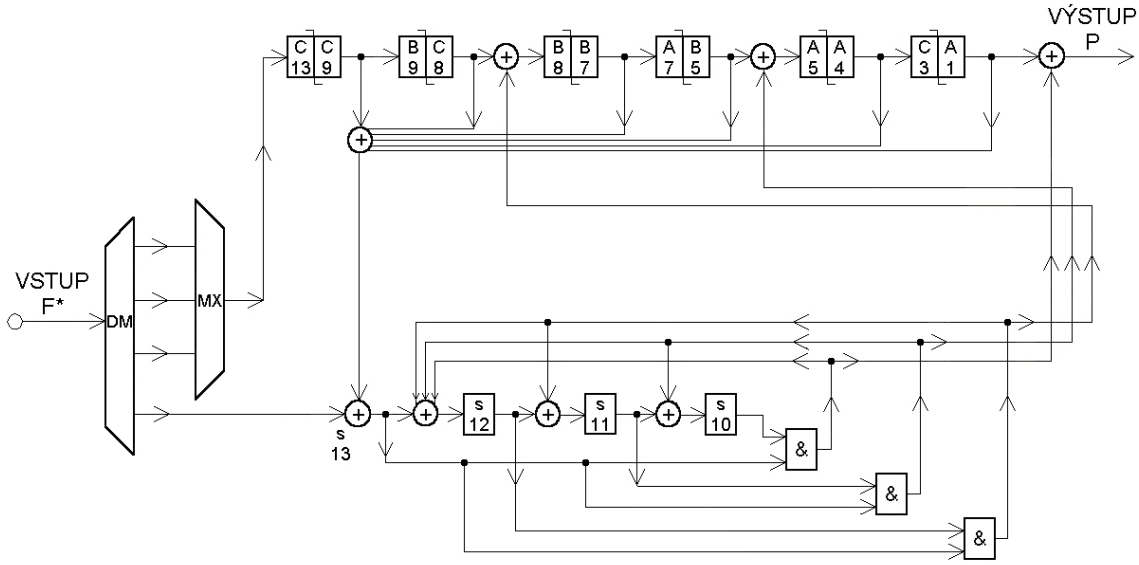
kde pro korekci jednotlivých chybných bitů platí:

$$s_{13} \cdot s_{12} = 0 \wedge s_{13} \cdot s_{12} = 1, \tag{4.37}$$

$$s_{13} \cdot s_{11} = 0 \wedge s_{13} \cdot s_{11} = 1, \tag{4.38}$$

$$s_{13} \cdot s_{10} = 0 \wedge s_{13} \cdot s_{10} = 1, \tag{4.39}$$

a výsledek je roven nule je-li konkrétní bit správný, při chybě je součin syndromů roven jedné, což indikuje nutnost provedení příslušné korekce.



Obr. 4.4: Dekodér Iwadari-Masseyho kódu pro korekci 4 chyb.

S využitím získaných výsledků je možné provést hlubší rozbor kodeku, tak jako v předchozím případě [46]. Základní zpoždění je způsobeno jen průchodem přes paměťové buňky v dekodéru, jelikož v kodéru slouží paměťové buňky k průběžnému tvoření zabezpečujících bitů. V dekodéru se paměťových buněk využívá k úschově přenesené informace. Při paralelním způsobu odvozování i přenosu, pro celkové zpoždění kodeku, které je dáno zpožděním dílčí paralelní větve, platí:

$$Z = Z_K + Z_D = Z_D = \frac{n_0 * (n_0 - 1)}{2} + (2n_0 - 1), \tag{4.40}$$

kde Z_K je zpoždění kodéru, Z_D zpoždění dekodéru. V případě sériového přenosu je zpoždění zřejmé v konkrétním případě z Obr. 4.4, kdy je dáno dílčí částí dekodéru – velikostí paměťového pole kodéru přijímače. Při využití (4.28) a (4.29) platí vztah:

$$Z_S = m * k_0 = 0{,}5n_0^3 + n_0^2 - 2{,}5n_0 + 1. \tag{4.41}$$

Při stanovení konstrukční složitosti se využívají znalosti rozměrů blokové matice $\mathbf{B_0}$ - stejně jako u Hagelbargerova kódu k vyjádření vztahů potřebného množství paměťových buněk a logických operátorů. Pro počet paměťových buněk kodéru S_{PK} platí:

$$S_{PK} = m * k_0. \tag{4.42}$$

Dekodér je vhodné rozdělit na menší dílčí části. Korekce probíhá v části kodéru přijímače proto jsou paměťové buňky u dekodéru využity pouze u kodéru přijímače S_{PD1} a přípravy korekce S_{PD3}:

$$S_{PD1} = S_{PK} = m * k_0, \tag{4.43}$$

$$S_{PD3} = n_0 - 1, \tag{4.44}$$

součet jednotlivých oblastí udává počet paměťových buněk dekodéru:

$$S_{PD} = S_{PD1} + S_{PD2} = (m * k_0) + (n_0 - 1). \tag{4.45}$$

Pro celkový počet paměťových buněk kodeku při použití vztahů (4.28) a (4.29) se dostane výraz:

$$S_P = S_{PK} + S_{PD} = 2(m * k_0) + n_0 - 1 = n_0^3 + 2n_0^2 - 4n_0 + 1. \tag{4.46}$$

Zbývá určit množství potřebných logických operátorů. V kodéru jejich počet určuje S_{LK} a platí:

$$S_{LK} = 1. \tag{4.47}$$

Rozčlenit dekodér je možné opět na tři části – kodér přijímače S_{LD1}, korekce S_{LD2} a příprava korekce S_{LD3}:

$$S_{LD1} = S_{LK} = 1, \tag{4.48}$$

$$S_{LD2} = n_0 - 1, \tag{4.49}$$

$$S_{LD3} = 2(n_0 - 1) + 1, \tag{4.50}$$

součet částí udává počet logických operátorů dekodéru:

$$S_{LD} = S_{LD1} + S_{LD2} + S_{LD3} = 3(n_0 - 1) + 2\,, \tag{4.51}$$

celý kodek pak obsahuje:

$$S_L = S_{LK} + S_{LD} = 3n_0\,. \tag{4.52}$$

4.4 Základní Berlekamp-Preparatův kód (n_0; n_0 -1; m)

Postup objevený nezávisle Berlekampem a Preparatem, vždy přinese kód splňující Gallagerovy meze. Gallager dokázal, že libovolný konvoluční kód o informační rychlosti R má schopnost opravit všechny shluky délky b nebo kratší vzhledem k ochrannému intervalu délky A, jestliže platí [39]:

$$\frac{A}{b} \geq \frac{1+R}{1-R}\,. \tag{4.53}$$

Zmíněný vztah (4.53) je znám jako mez kompletní opravy shlukové chyby. Z tohoto důvodu jsou Berlekamp-Preparatovy kódy optimální pro opravu postupných shlukových chyb.

Jedná se o systematický konvoluční kód ke korekci shluků omezených do samostatného bloku úměrného ochrannému intervalu m bezchybných bloků (tj. shluk může ovlivnit nanejvýš jeden blok omezené délky). Tento kód by měl mít schopnost postupné opravy shlukových chyb jednoho bloku úměrně ochrannému intervalu m bloků. Opět se k základnímu popisu používá vytvářecí matice $\mathbf{B_0}$, jejíž rozměry jsou n_0 x $2n_0$, jenž se skládá ze dvou podmatic [53]. První podmatice má jedničky na vedlejší diagonále a druhá podmatice má jedničky umístěné nad hlavní diagonálou. Přičemž obě podmatice mají shodný rozměr n_0 x n_0. Zbylé prvky podmatic jsou nulové viz. vztah (4.54):

$$\mathbf{B_0} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.54}$$

a obecná bloková matice $\mathbf{B_0}$ má rozměry:

$$\mathbf{B_0}[n_0; n_0 - 1; m] = (n_0; 2n_0)\,. \tag{4.55}$$

Ze vztahu (4.53) a (4.2) při podmínce $R = (n_0 - 1)/n_0$ se získají další důležité parametry příslušného kódu:

$$A = m * n_0 = m * b\,, \tag{4.56}$$

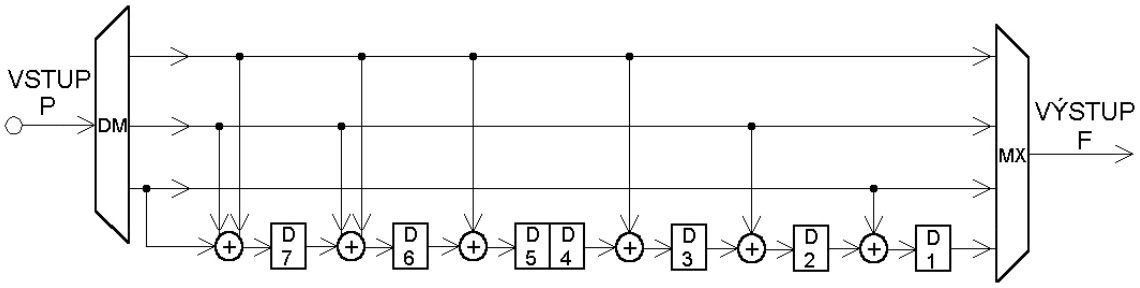
$$\frac{A}{b} = m = 2n_0 - 1. \tag{4.57}$$

Bloková vytvářecí matice $\mathbf{B_0}$ stanovuje způsob zapojení kodéru viz. Obr. 4.5 – určují jej zabezpečovací vytvářecí mnohočleny. Jedná se o systematický konvoluční kód s generačními polynomy vyplývající z $\mathbf{B_0}$:

$$g_1^{(4)} = D^3 + D^5 + D^6 + D^7, \tag{4.58}$$

$$g_2^{(4)} = D^2 + D^6 + D^7, \tag{4.59}$$

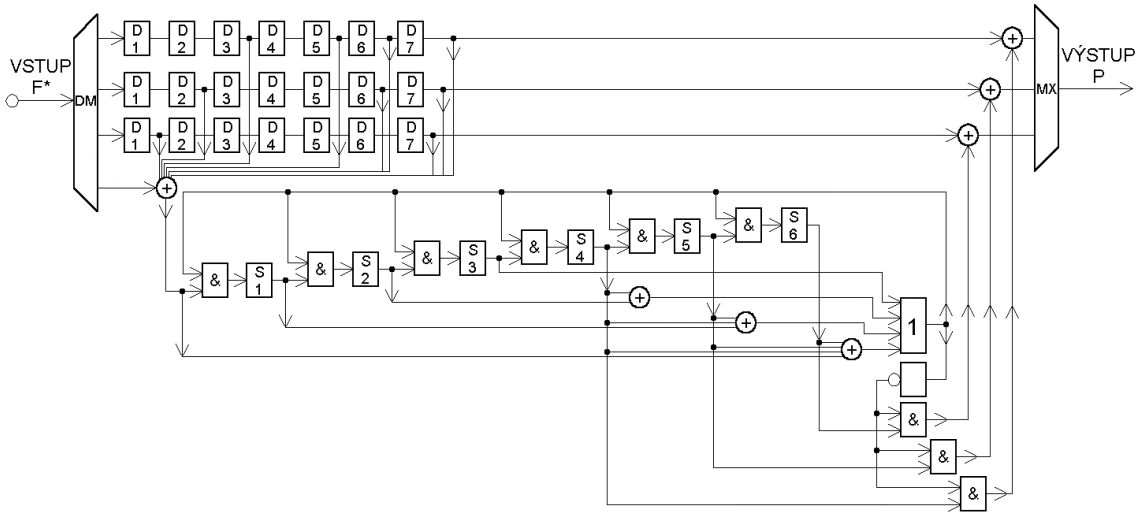
$$g_3^{(4)} = D + D^7. \tag{4.60}$$



Obr. 4.5: Kodér Berlekamp-Preparatova kódu pro korekci 4 chyb.

Berlekamp-Preparatovy kódy mohou být dekódovány použitím obecné dekódovací techniky pro konvoluční kódy opravující shlukové chyby zásluhou Masseyho [54]. V uvedeném dekodéru nemůže nastat nekonečné šíření chyby a jeho schéma je uvedeno na Obr. 4.6. Vychází se z kontrolní matice $\mathbf{H_0}$, jenž udává zapojení dekodéru pro syndromové dekódování:

$$\mathbf{H_0} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{4.61}$$



Obr. 4.6: Dekodér Berlekamp-Preparatova kódu pro korekci 4 chyb.

K dispozici jsou známy hodnoty, s nimiž lze dále pracovat na podrobnějším popisu kodeku. Základní zpoždění je způsobeno jen průchodem přes jednu dílčí část paralelního větvení paměťových buněk v dekodéru, jež slouží k vytvoření kodéru přijímače a využívá se jich pro úschovu přenesené informace za účelem případné korekce chyb. V kodéru se zabezpečení tvoří průběžně a tedy neovlivňuje přenášenou informaci. Pro základní zpoždění Berlekamp-Preparatova kodeku platí vztah:

$$Z = Z_K + Z_D = Z_D = 2n_0 - 1, \tag{4.62}$$

kde Z_K je zpoždění kodéru a Z_D zpoždění dekodéru. V případě sériového přenosu informace se v dekodéru uplatní zpoždění všech informačních větví:

$$Z_S = Z * (n_0 - 1) = 2n_0^2 - 3n_0 + 1. \tag{4.63}$$

Jako v dřívějších případech se využije známosti rozměrů blokové matice $\mathbf{B_0}$ pro stanovení konstrukční složitosti jednotlivých celků. Pro počet paměťových buněk kodéru S_{PK} platí:

$$S_{PK} = 2n_0 - 1. \tag{4.64}$$

Dekodér se nejdříve rozdělí na jednotlivé logické celky. Případná korekce probíhá v rámci části kodéru přijímače, i proto jsou paměťové buňky u dekodéru využity pouze u kodéru přijímače S_{PD1} a přípravy korekce S_{PD3}:

$$S_{PD1} = (n_0 - 1) * S_{PK} = (n_0 - 1) * (2n_0 - 1) = 2n_0^2 - 3n_0 + 1, \tag{4.65}$$

$$S_{PD3} = 2(n_0 - 1), \tag{4.66}$$

sečtením se obdrží hodnota za dekodér:

$$S_{PD} = S_{PD1} + S_{PD3} = 2n_0^2 - n_0 - 1. \tag{4.67}$$

Pro počet paměťových buněk celého kodeku pak platí vztah:

$$S_P = S_{PK} + S_{PD} = 2n_0^2 + n_0 - 2. \tag{4.68}$$

Závěrem pro počet logických operátorů v kodéru S_{LK} platí:

$$S_{LK} = 2(n_0 - 1). \tag{4.69}$$

U dekodéru jsou využity logické operátory jen u korekce S_{LD2} a přípravy korekce S_{LD3}, jelikož z vlastního kodéru přijímače jsou vyvedeny jen vazby do dalších dílčích částí:

$$S_{LD2} = n_0 - 1, \tag{4.70}$$

$$S_{LD3} = 4n_0 - 1, \tag{4.71}$$

součtem dílčích částí pak pro dekodér platí:

$$S_{LD} = S_{LD2} + S_{LD32} = 5n_0 - 2, \tag{4.72}$$

pro celý kodek:

$$S_L = S_{LK} + S_{LD} = 7n_0 - 4. \tag{4.73}$$

4.5 Srovnání jednotlivých variant

Na základě vytvořeného rozšířeného matematického aparátu, jenž popisuje jednotlivé vlastnosti kódů lze snadněji pro konkrétní případy provést jasné srovnání. Nově odvozené vztahy s využitím blokové vytvářecí matice mají všeobecnou platnost. Nejprve bude uvedeno porovnání konvolučních kódů pro opravu shlukových chyb, jejichž podrobný popis i jednotlivé vztahy jsou uvedeny výše. Následně se provede i konfrontace s jinými nejčastěji používanými způsoby ochrany zprávy před shlukem chyb.

4.5.1 Skupina konvolučních kódů

Vlastnosti prezentovaných konvolučních kódů, jsou uvedeny v Tab. 4.1. Hodnoty dalších vlastností konvolučních kódů v tabulce jsou uvedeny pro maximální opravitelnou délku shluku chyb $b \leq 4$ pomocí jednotlivých vztahů z kapitoly 4.2, 4.3 a 4.4. Dalším společným znakem všech kódů je stejná informační rychlost R, jenž ze vztahu (4.2) odpovídá hodnotě $R = 0{,}75$.

Tab. 4.1: Konvoluční kódy s hodnotami sledovaných vlastností.

Kód	A [bit]	Z [bit]	S_P [-]	S_L [-]
Hagelbargerův	47	11	55	23
Iwadari-Masseyho	51	13	81	12
Berlekamp-Preparatův	28	7	34	24

Z tabulky je zřejmé, že při stejné informační rychlosti a velikosti opravitelné chyby má sledované veličiny nejnižší (a tím pádem nejlepší vlastnosti) Berlekamp-Preparatův kód. Vyžaduje nejkratší ochranný interval, způsobuje nejkratší zpoždění přenášené zprávy průchodem paměťových buněk a k realizaci kódu je třeba nejmenší počet paměťových buněk. Navíc na rozdíl od Hagelbargerova kódu nemůže u Berlekamp-Preparatova a Iwadari-Masseyho kódu nastat nekonečné šíření chyby. Uvedené vlastnosti jsou vykoupeny největším počtem potřebných logických operátorů k zajištění činnosti kodeku. Však rozdíl není tak dramatický jako u počtu paměťových buněk. Nejméně logických funkcí pro správné zajištění funkčnosti kodeku dle Tab. 4.1 představuje Iwadari-Masseyho kód.

Výše popsaný způsob pracuje s kódovým zabezpečením v podstatně širších souvislostech. Neomezuje se pouze na zabezpečovací kód a všímá si dalších vlastností, které jsou důležité pro výběr optimální varianty při realizaci [45], [46].

4.5.2 Skupina používaných systémů

K podpoře argumentu, že existuje mezera používaných systémů pro korekci shluků chyb různé délky mezi základními blokovými kódy na straně jedné a sofistikovanými systémy založené na Turbo-kódech či Reed-Solomonových kódech na straně druhé, je níže provedeno srovnání zástupců uvedených kategorií se zástupcem konvolučních kódů pro opravu shlukových chyb. U individuálních protichybových systémů, kde je vyžadováno při konstrukci především splnění všech protichůdných podmínek (viz kapitola 3) a neklade se důraz na masovou výrobu, lze pak v některých oblastech dosáhnout lepších výsledků [4], [16], [57].

Výběr je proveden s ohledem na základní vlastnosti a parametry kódů, jenž se uplatňují u všech tří zkoumaných skupin – dosahovaná informační rychlost, minimální bezchybná vzdálenost před dalším shlukem chyb a délka minimálního zpoždění daná zpracováním potřebného bloku či provedení korekce. Stručná charakteristika jednotlivých kódů:

Hammingovy kódy:

- Velice jednoduchý kód.
- Nenáročné kódování a dekódování.

- Opravuje pouze jednu chybu.
- Použitelný pouze v kombinaci s prokládáním.

Reed-Solomonovy kódy:

- Opravují shlukové chyby.
- Opravuje celé symboly bez ohledu na počet chyb v symbolu.
- Složitý proces kódování a dekódování signálu.
- Hojně se také využívají v kombinaci s prokládáním či zřetězením.

Berelekamp-Preparatovy kódy:

- Opravují shlukové chyby.
- Proces kódování a dekódování není příliš komplikovaný.
- Možnost vygenerování kódu na přesně požadovanou délku shluku chyb.
- Prokládání je možné, avšak nevyužívá se.

Konfrontace nejčastěji používaných protichybových systémů [1], [4] se systémem konvolučního kódování bude provedena pro shluk chyb délky $b \leq 5$ bitů. Lze se domnívat, že pro klasické blokové kódy velikost této chyby již představuje zvýšené nároky na zabezpečení jednoho bloku a dochází tak ke snížení dosahované informační rychlosti. Při nasazení robustních systémů sloužících k opravě dlouhých shluků chyb se stále ještě jedná o plýtvání jejich výpočetním výkonem, jelikož využívají komplikovaných výpočetních algoritmů jak pro kódování tak především pro dekódování.

Hammingův kód je vybrán pro svoji jednoduchost a velmi nenáročný způsob kódování i dekódování. Což jsou vlastnosti, které mu při hledání nejlepších variant řešení opravy náhodných chyb poskytují vysoké ohodnocení [7], [16]. Jelikož Hammingův kód je schopen opravit pouze jedinou chybu je třeba pro opravu chybného úseku 5 bitů využít navíc systém založený na bitovém prokládání, jenž zaručí rozložení shluku chyb do jednotlivých chyb v každém bloku.

Hammingův kód (7, 4) opravuje jeden chybný bit v kódové kombinaci délky *n* bitů. Podle rovnice (2.3) vyplývají pro rozměry požadované prokládací matice, pro rozklad shluku chyb 5 bitů, následující parametry:

$$j \geq \frac{b}{t} \geq \frac{5}{1} \geq 5,$$

pak velikost prokládací matice je dána vztahem (2.5):

$$PM = j * n = 5 * 7,$$

$$PM = 35 bitů.$$

Kódové slovo má délku n = 7 bitů. Na základě této hodnoty se určí minimální délka ochranného intervalu A, dle (2.4):

$$A \geq n * j - b \geq 7 * 5 - 5,$$
$$A \geq 30 bitů.$$

Pro informační rychlost R z (4.2) platí:

$$R = \frac{k}{n} = \frac{4}{7} = 0{,}57 .$$

Minimální zpoždění systému je dáno průchodem dvou prokládacích matic a velikosti informace ve zpracovávaném bloku.

$$Z_S = 2 * PM + k = 2 * j * n + k = 2 * 5 * 7 + 4,$$
$$Z_S = 74 bitů.$$

Na stejný požadavek korekce shluku chyb $b \leq 5$ bitů se v druhém případě uplatní vícestavové kódy používané pro zabezpečení proti shlukům chyb – RS kódy. V tomto případě nebude nutné použít blok prokládání, jenž by jen zvyšoval celkové zpoždění. Jen počet opravovaných symbolů v daném bloku musí pokrýt počet chybných bitů – při libovolném umístění shlukové chyby. Při dodržení podmínky (2.2) v případě použití čtyřstavových bitových symbolů dle vztahu (2.1) platí následující:

$$n = 2^m - 1 = 2^4 - 1,$$
$$n = 15.$$

Pro počet chybných symbolů v kódu:

$$t \geq \frac{b}{m} \geq \frac{5}{4},$$
$$t = 2.$$

Tedy jeden shluk chyb poškodí dva čtyřstavové symboly bez ohledu na bitové poloze daného shluku. Následně zbývá určit počet zabezpečovacích symbolů z (2.1):

$r = 2t = 4$,

kde počet informačních symbolů je roven z (2.1):

$k = n - r = 15 - 4 = 11$.

Pak je definován kód RS (15, 11), který má informační rychlost R z (4.2):

$$R = \frac{k}{n} = \frac{11}{15} = 0{,}73 .$$

Shluk 5 chyb se může vyskytnout pouze jednou v daném bloku kódu RS (15, 11), aby jej bylo možné korigovat a s tím souvisí i délka ochranného intervalu A, jenž je určena v nejnepříznivější variantě velikostí daného bloku, jak informační tak zabezpečující části, v bitech:

$$A \geq n * m - b \geq 15 * 4 - 5,$$
$$A \geq 55 bitů.$$

Minimální zpoždění systému je dáno zpožděním doručení celého jednoho zpracovávaného informačního bloku, přičemž rychlost v přenosovém kanálu musí být vyšší než na vstupu-výstupu kodeku. Pak platí:

$$Z_S = m * k = 4 * 11,$$
$$Z_S = 44 bitů.$$

Na závěr se použije Berlekamp-Preparatův kód, což je zástupce konvolučních kódů pro opravu shlukových chyb. Ani v tomto případě nebude nutné použít blok prokládání, jenž by jen zvyšoval celkové zpoždění systému a navíc je možné navrhnout systém přesně na opravu daného shluku chyb. Dle vztahu (4.57):

$$b = n_0 = 5,$$
$$m = 2n_0 - 1 = 10 - 1 = 9.$$

Pak je definován Berlekamp-Preparatův kód (5, 4, 9), který má informační rychlost R z (4.2):

$$R = \frac{k_0}{n_0} = \frac{4}{5} = 0{,}80.$$

Délka ochranného intervalu A je jedním ze základních parametrů této skupiny kódů a pro její velikost platí (4.56):

$$A \geq m * n_0 \geq 9 * 5,$$
$$A \geq 45 bitů.$$

Minimální zpoždění systému při sériovém způsobu přenosu je dáno z předcházející části odvozeným vztahem (4.63):

$$Z_S = 2n_0^2 - 3n_0 + 1 = 2 * 5^2 + 3 * 5 + 1,$$
$$Z_S = 36 bitů.$$

Všechny vypočítané vlastnosti představených zástupců jednotlivých používaných systémů pro korekci shluků chyb přehledně udává souhrnná Tab. 4.2. Výsledky potvrzují, že v individuálních protichybových systémech lze dosáhnout lepších parametrů, využitím skupiny konvolučních kódů pro opravu shluků chyb konkrétní velikosti.

Tab. 4.2: Hodnoty základních vlastností srovnávaných kódů pro $b \leq 5$.

Kód	A [bit]	Z_S [bit]	R [-]
Hammingův (7, 4)	30	74	0,57
Reed-Solomonův (15, 11)	55	44	0,73
Berlekamp-Preparatův (5, 4, 9)	45	36	0,80

5 Simulace

Pro prověření správnosti konstrukce schémat a zabezpečovacích schopností jednotlivých konvolučních kódů jsou získané výsledky podrobeny kontrole pomocí simulace. Simulace představuje základní metodu pro ověření i prezentování již navrženého zabezpečovacího procesu. To umožní snazší pochopení dané značně rozsáhlé problematiky. Základní požadavky kladené na program jsou:

- stabilita,
- rozsáhlá knihovna prvků,
- uživatelsky přívětivé grafické prostředí,
- jednoduché a intuitivní ovládání,
- analýza případných chyb,
- podrobná dokumentace.

Programů pro simulaci je možno nalézt celou řadu - WinSpice, Micro-Cap, nadstavba Matlabu Simulink. Proto byl další z požadavků hlavně kladen na jednoduché a intuitivní ovládání. Za vhodnou volbu pro kódovací proces byl zvolen matematický program Matlab [14], [48], [52]. Především lze využít specializované knihovny Simulink, jenž slouží k modelování a simulaci dynamických systémů. Graficky zadávaná soustava je složena z bloků, které jsou vybírány z různých knihoven. Prostředí Simulinku potom umožňuje graficky sledovat průběhy veličin v libovolném místě zapojení.

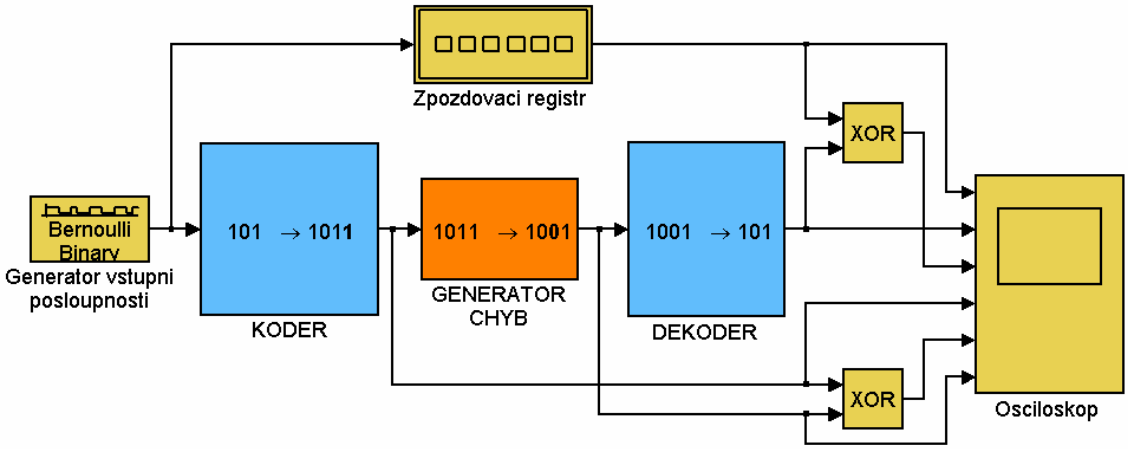
Ve zmiňovaném prostředí je možnost z jednotlivých bloků přímo z knihoven poskládat požadovaný funkční model. Pro jednotlivé kódy v rámci zkoumání jejich vlastností je vhodné též vypracovat kompletní modely kodeků. Modely dynamických soustav se vytvářejí interaktivně ve formě blokových schémat a propojení mezi nimi. Simulink je vytvořen pro časové řešení - simulací chování dynamického systému, pokud známe jeho matematický popis. Umožňuje sledovat veličiny téměř v jakékoliv části vzniklého modelu. Simulink obsahuje mnoho definovaných bloků, jak lineárních tak i nelineárních, už ve standardních knihovnách.

Podstatnou výhodu uvedený program představuje v možnostech kvalitního grafického znázornění, jenž je důležité pro seznámení s vytvářením a funkcí zabezpečovacích kódů vzhledem k jejich velké složitosti. Tyto kvalitnější prezentační metody zabezpečovacích korekčních kódů, kromě větší názornosti umožňují i zlepšit pohled na danou problematiku. Existují i další simulační prostředky, avšak intuitivní ovládání, názornost i přehlednost jejich výsledků je nižší než u vybraného matematického programu Matlab [17].

5.1 Popis simulačního zapojení modelu kodeku

Na Obr. 5.1 je uvedeno obecné blokové schéma zapojení daného simulovaného kodeku. Kromě bloku kodér a dekodér příslušného použitého kódu, jsou ještě použity bloky generátor vstupního toku dat, přenosový kanál (kde je umístěn generátor chyb) a zobrazení výstupního toku dat. Konkrétní schémata návrhu kodeků byla demonstrována pro opravu shluků 4 chyb. Pro přehlednost je simulace dělána pro tyto konkrétní schémata, ovšem vlastní popis je co nejobecnější.

Bernoulliho binární generátor vytváří sériový nezabezpečený bitový tok, který jde na vstup demultiplexoru zvoleného kodéru. Na výstupu demultiplexoru kodéru jsou dílčí paralelní toky, jejichž počet je dán požadovaným zabezpečením, jenž nezměněny projdou vlastním kodérem. Na výstupu příslušného kodéru se k nim přidá zabezpečující bitový tok, který byl v kodéru vytvořen. Všechny tyto bitové toky tvoří vstupy multiplexoru kodéru. Na výstupu multiplexoru kodéru je zabezpečený sériový bitový tok, jenž projde generátorem chyb, což představuje chybový přenosový kanál, který zavádí do přenášené posloupnosti shluky chyb. Na vstup demultiplexoru dekodéru se dostává zabezpečený sériový bitový tok se shluky chyb, jenž se rozdělí na paralelní bitové toky. Uvedené bitové toky tvoří vstupy vlastního dekodéru, kde se provádí případná korekce chyb a na výstupu dostáváme původní dílčí paralelní bitové toky před přenosem. Na výstupu multiplexoru v dekodéru je původní, avšak zpožděný sériový bitový tok, jenž je vytvářen Bernoulliho binárním generátorem.



Obr. 5.1: Obecné blokové schéma zapojení kodeku.

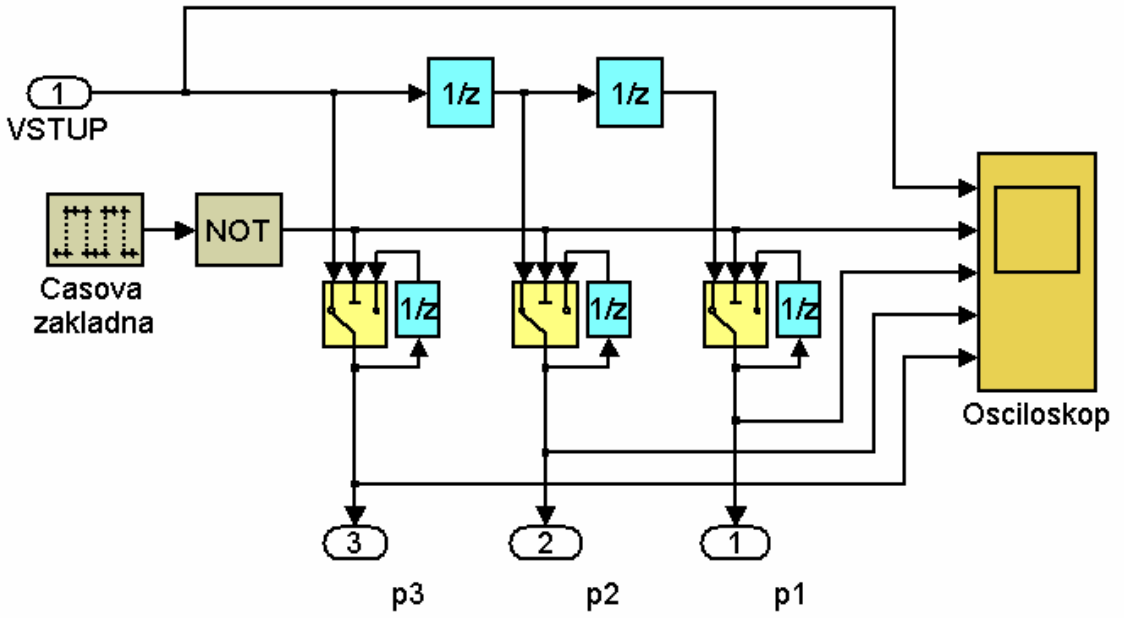
Na Obr. 5.2 je zobrazeno obecné vnitřní zapojení bloku kodéru pro korekci shluku chyb $b \leq 4$. Je zřejmé, že se skládá ze tří dílčích podbloků. Prvním z nich je sérioparalelní převodník, který v uvedeném příkladě převádí vstupní sériová data na tři paralelní toky dat pro kodér. Ve vlastním kodéru jsou vstupní data vybavena zabezpečovacím bitem dle pravidel daného kódu a pokračují do bloku paralelně sériového převodníku, který převádí paralelní data zpět na sériový tok vhodný pro vysílání přenosovým kanálem. Pro sériově-paralelní a paralelně sériové převodníky nelze použít bloky z knihovny Commonly Used Blocks – Mux, Demux, jelikož nejsou

pro zadanou simulaci vhodné, protože u nich nelze nastavit dobu trvání jednoho vzorku vstupního a výstupního signálu [55].

Obr. 5.2: Obecné blokové schéma zapojení kodéru s úpravou rychlosti při $b \leq 4$.

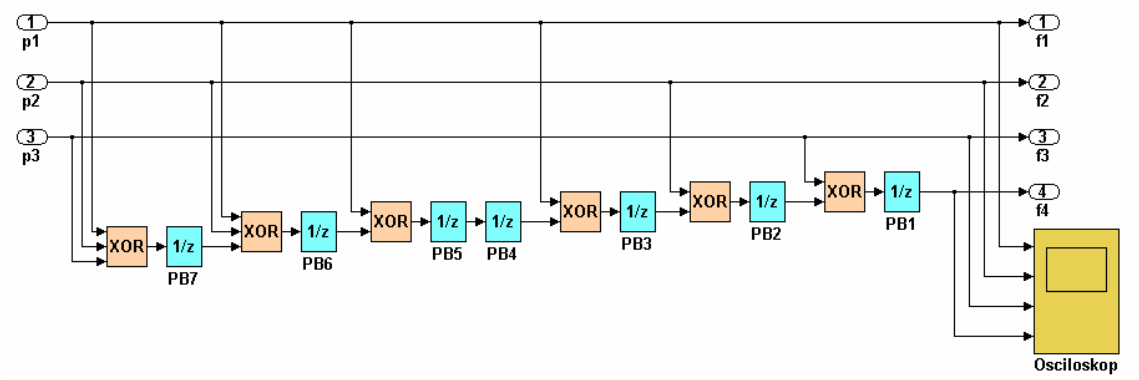
Jak je patrné z Obr. 5.2, v zapojení bloku kodéru se vyskytují dvě různé přenosové rychlosti. Do bloku vlastního kodéru vstupují tři paralelní bity převedené sérioparalelním převodníkem ze vstupních sériových dat o přenosové rychlosti v_1, na výstupu však získáváme paralelní bity čtyři, které paralelně sériový převodník převádí na sériová data o přenosové rychlosti v_2. Pro přenosové rychlosti tedy platí poměr $v_1 / v_2 = 3/4$.

Vnitřní schéma sériově paralelního převodníku je zobrazeno na Obr. 5.3. Vstupní data jsou přiváděna rychlostí v_1 do posuvného registru tvořeného dvěma paměťovými buňkami. S příchodem třetího signálového prvku na vstup převodníku jsou pomocí impulzu z časové základny přepnuty řízené spínače a dojde k odběru vzorku logických hodnot obsahu paměťových buněk. Tento odebraný vzorek je po zpětném přepnutí spínačů uchován ve smyčce tvořené výstupem a druhým vstupem spínače až do doby než dojde k odběru následujícího vzorku. Celé zapojení tak plní funkci demultiplexoru.



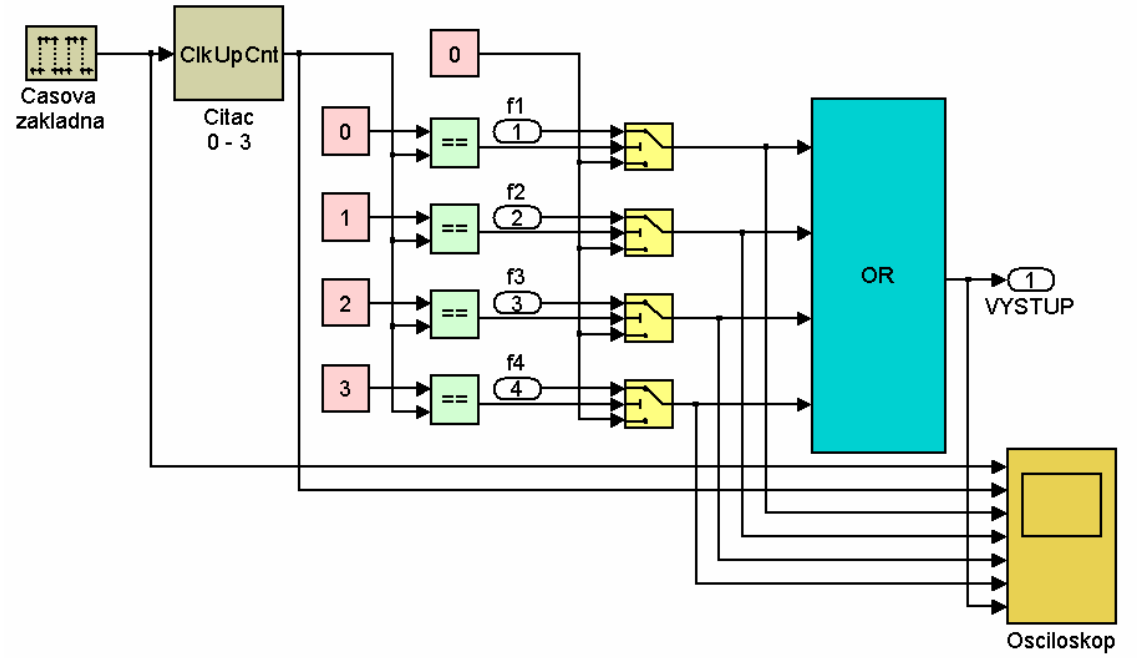
Obr. 5.3: Obecný sériově paralelní převodník při $b \leq 4$.

Vnitřní zapojení bloku vlastního kodéru je určeno vytvářecí maticí, každého kódu. Jako příklad je uveden Berlekamp – Preparatův kodér, jenž je schopen korigovat shluk chyb $b \leq 4$. Na vstup kodéru jsou přiváděny tři dílčí paralelní signálové toky, ze kterých jsou odebírány vzorky logických hodnot viz. Obr 5.4, jenž realizuje provedení zabezpečení z Obr 4.5. Pomocí součtů mod2, reprezentovaných bloky XOR (Exklusive OR), a sedmi paměťových buněk, reprezentovaných zpožďovacími členy 1/z, je vytvářen zabezpečovací bit pro danou trojici vstupních bitů.



Obr. 5.4: Berlekamp – Preparatův kodér, realizace v Matlabu.

Blok paralelně sériového převodníku má za úkol převést čtyři výstupní bity z vlastního kodéru na sériový tok bitů, o rychlosti v_2, vhodný pro přenos. Jeho vnitřní zapojení ukazuje Obr. 5.5.



Obr. 5.5: Obecný paralelně sériový převodník při $b \leq 4$.

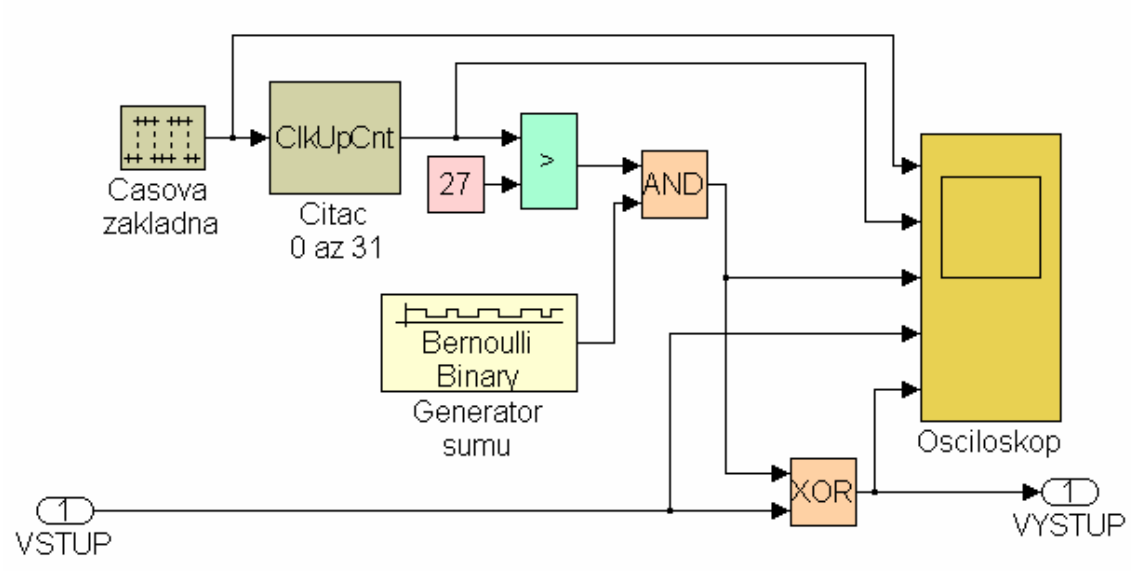
Funkce převodníku je založena na postupném vyčítání vzorků signálů přivedených na vstupy f1 - f4. Aktuálně vyčítaný vstup je určen přepnutím příslušného spínače řízeného pomocí komparátoru. Komparátor porovnává hodnotu z čítače 0 - 3

řízeného časovou základnou pracující rychlostí v_2 s hodnotou konstanty příslušné pro každý vstupní tok. Při shodě hodnoty konstanty a hodnoty z čítače je přepnut daný spínač a vzorek je přiveden na sčítací hradlo. Výstupní tok bitů o rychlosti v_2 se získá logickým součtem dílčích toků pomocí čtyř vstupového hradla OR. Díky tomu jsou dílčí vstupní toky multiplexovány do jednoho toku výstupního.

Veškerá upravená - zabezpečená data jsou vyslána po přenosovém kanálu, kde může dojít k poškození. Na práci v binární soustavě je založen celý aparát detekce i korekce vzniklých chyb. Digitální přenos představuje posloupnost bitů, což jsou úseky datového přenosu o konstantní délce a určité úrovni, které v případě použití binárního kódu mají dvě hodnoty, a to 0 nebo 1. Z toho vyplývá i způsob vzniku chyb při přenosu takové posloupnosti nul a jedniček. Tedy k chybě v datovém kanálu může dojít různými vlivy pouze změnou hodnoty bitu, buď z 0 na 1 nebo z 1 na 0.

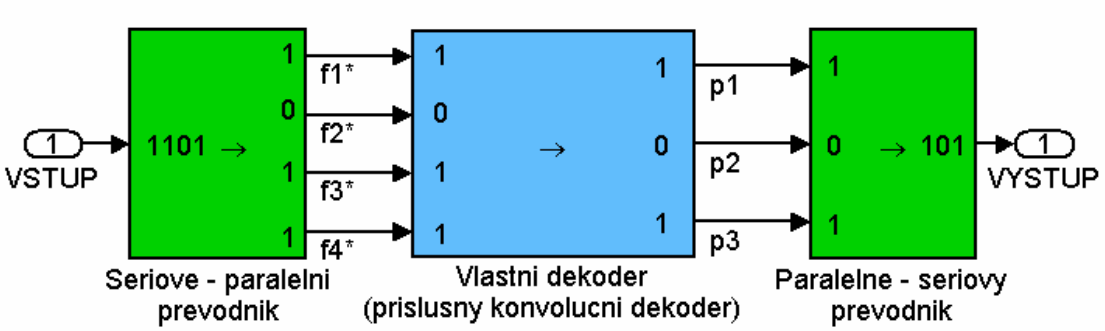
Blok generátor chyb simuluje shlukové chyby, jenž mohou vznikat v reálném přenosovém kanálu. Chybu lze modelovat jednoduše sčítačkou mod2, kdy je přičítán chybový vektor k přenášené zprávě nebo sofistikovaněji blokem generátoru chyb, který je znázorněn na Obr 5.6. Lze tak jednoduše ovlivňovat možnou délku shlukové chyby a vzdálenost mezi chybami.

Komparátor srovnává hodnoty výstupu čítače se zvolenou konstantou. V případě splnění podmínky (hodnota na výstupu čítače je větší než daná konstanta) výstup komparátoru se nastaví na log. 1. Konstanta se nastavuje na příslušnou hodnotu dle požadované maximální délky shluku chyb: rozdíl mezi maximální hodnotou čítače a délkou shluku chyb. V uvedeném příkladě tedy 31 – 4 = 27, což je hodnota dané konstanty. Právě log. 1 z výstupu komparátoru zajistí, že náhodně generované bity z Bernoulliho binárního generátoru (shluky chyb) se dostanou na výstup logického členu AND a přes logický člen XOR jsou tyto bity přidány do sériového zabezpečeného bitového toku. Tímto krokem dojde k přidání náhodného shluku chyb do sériového zabezpečeného toku.



Obr. 5.6: Generátor shlukových chyb.

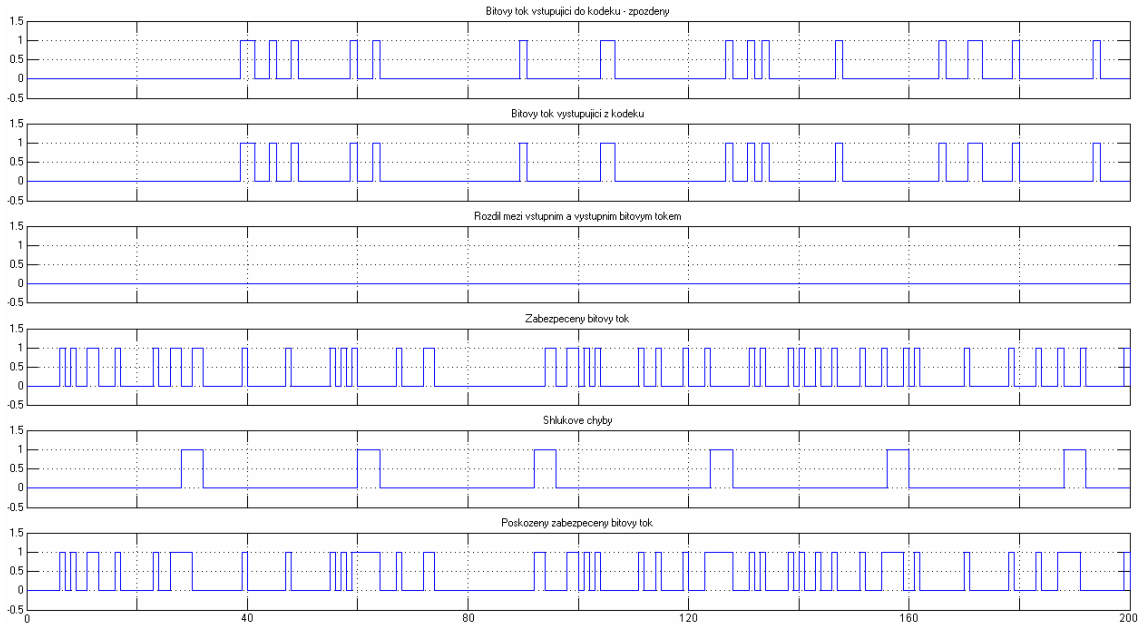
Dekodér je opět složen ze tří podbloků viz. Obr. 5.7. Sériově paralelní převodník pro daný příklad převádí vstupní data o rychlosti v_2 na čtyři paralelní dílčí toky. Ve vlastním dekodéru proběhne oprava chyb na základě syndromů získaných ze zabezpečovacích bitů. Sčítačkami mod2 se k signálovým prvkům přičítají případné příznaky chyb, čímž se tyto chyby eliminují a na výstupu dekodéru jsou již data opravená. Příznaky chyb se vytváří v dílčí částí dekodéru výběrem určitých vzorků ze vstupních dat. Výběr vzorků vstupních dat z posuvných registrů je dán kontrolní maticí $\mathbf{H_0}$ daného použitého kódu.



Obr. 5.7: Obecné blokové schéma zapojení dekodéru s úpravou rychlosti při $b \leq 4$.

Vnitřní schéma zapojení dekodérů odpovídá příslušnému použitému konvolučnímu kódu viz. Obr 4.2, Obr. 4.4 a Obr. 4.6. Jednotlivé opravené dílčí toky jsou na závěr pomocí paralelně sériového převodníku multiplexovány do výstupního bitového toku o rychlosti v_1. Základní vnitřní zapojení multiplexoru a demultiplexoru je shodné s kodérem viz Obr. 5.3 a Obr. 5.5, liší se pouze počet zpracovávaných paralelních toků [55].

Propojením všech dílčích částí lze provést kompletní simulaci kodeku daného modelu s grafickým výstupem viz. Obr. 5.8. Vyobrazeny jsou následující grafy kodeku Berlekamp – Preparatova kódu pro opravu shluků chyb $b \leq 4$. Na prvním z nich je zobrazena vstupní nezabezpečená posloupnost bitů o přenosové rychlosti v_1, která je ovšem zpožděná. Zpoždění je zavedeno pro lepší přehlednost grafu (odpovídající signálové prvky jsou pod sebou). Na druhém grafu je zobrazena výstupní dekódovaná posloupnost bitů o stejné přenosové rychlosti. Třetí graf znázorňuje rozdíl mezi vstupními a dekódovanými daty. Nulový průběh značí shodu mezi signály. Na čtvrtém grafu jsou znázorněna data zakódovaná kodérem, jenž vstupují do simulovaného přenosového kanálu přenosovou rychlostí v_2. Následující graf ukazuje shlukové chyby, které ovlivňují přenášený signál. V posledním grafu jsou přenášená data s vloženými shluky chyb, jenž vstupující do obvodu dekodéru. Jelikož je zpožděná vstupní datová posloupnost rovna výstupní (viz. třetí průběh) došlo tedy ke korekci vzniklých chyb.



Obr. 5.8: Grafický výstup simulace pro Berlekamp – Preparatův kód.

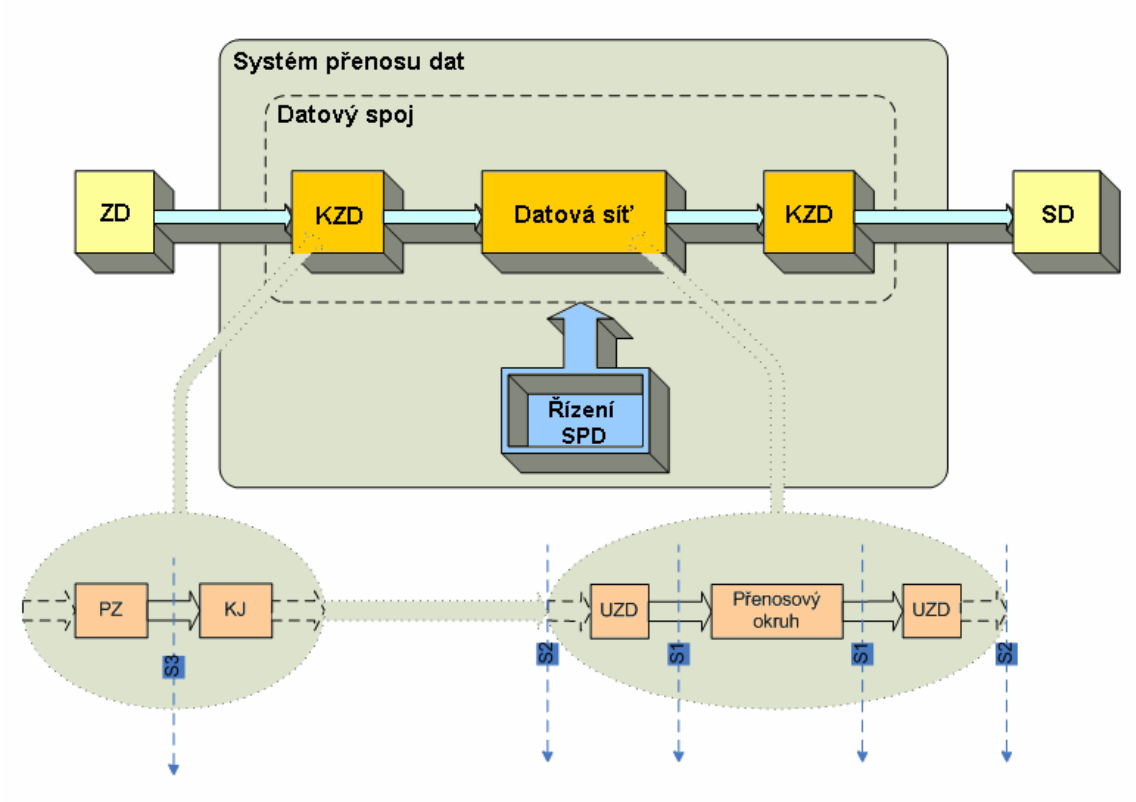
Získané výsledky simulace potvrzují správnou funkci příslušného kodeku. Navržená schémata zapojení z kapitoly 4.2, 4.3, 4.4 a rozšiřující kritéria popisu kódů odvozená i na základě blokové vytvářecí matice byla touto simulací dostatečně prověřena [51], [56]. Toto grafické zobrazení poskytuje názornější pohled na problematiku zabezpečovacích korekčních kódů. Na přiloženém CD se nacházejí další detailní modely představených kodeků v kapitole 4, jejichž popis činnosti byl zde uveden.

6 Realizace

Zbývá ověřit realizovatelnost individuálních protichybových systémů. V tomto kroku je třeba součinnost zařízení nejen pro bezprostřední realizaci kódu. Proto začátek bude věnován začleněním vlastního protichybového systému v přenosovém řetězci. Následně se přistoupí ke způsobům návrhu číslicových systémů.

6.1 Systém přenosu dat

Systém přenosu dat nebo-li SPD je soubor podrobně nespecifikovaných zařízení, jenž slouží k technické realizaci přenosu dat [24]. Obecné schéma systému je uvedeno na Obr. 6.1, kde je znázorněna cesta datového přenosu mezi zdrojem dat (ZD) a spotřebičem dat (SD). Z bloku ZD se data přenáší do koncového zařízení přenosu dat (KZD). Zmíněný blok lze dále rozdělit na periferní zařízení (PZ) a komunikační jednotku (KJ). Dále vstupují data do datové sítě tvořené přenosovým zařízením přenosu dat (UZD) a přenosovým okruhem.



Obr. 6.1: Blokové schéma systému přenosu dat.

UZD je umístěno nikoliv ve vzdálené síti, ale v objektu uživatele. Uvedené zařízení navazuje na přípojná vedení a zajišťuje funkce vytváření, udržování i rušení

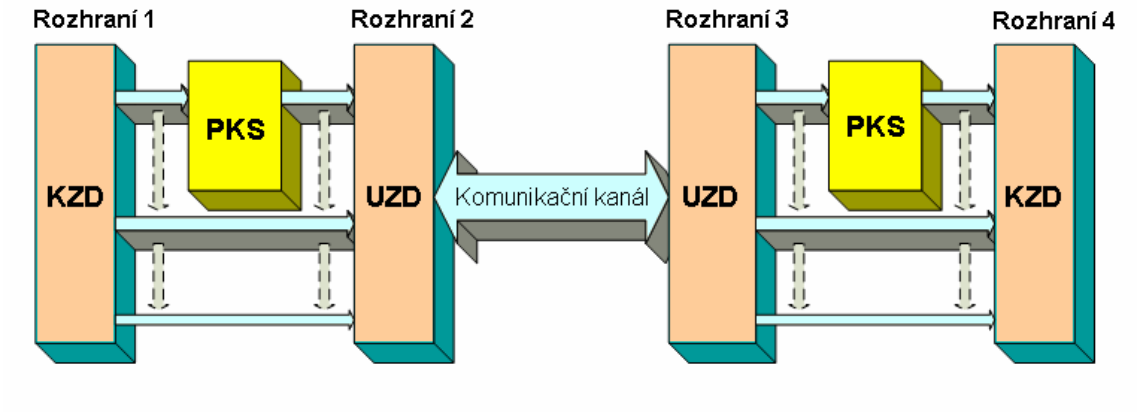
datového spoje a také funkce kódování i případné přeměny signálů mezi KZD. Zabezpečení přenosu dat se odehrává v komunikační jednotce.

Pro zajištění spolupráce a kompatibility různých zařízení od různých výrobců jsou mezi jednotlivými bloky definována rozhraní S_x, kde $x = \{1, 2, 3\}$. Tato rozhraní jsou v daném systému jasně definována a jejich elektrické, funkční, mechanické, i operační vlastnosti jsou předem přesně určeny.

6.2 Umístění protichybového kódového systému

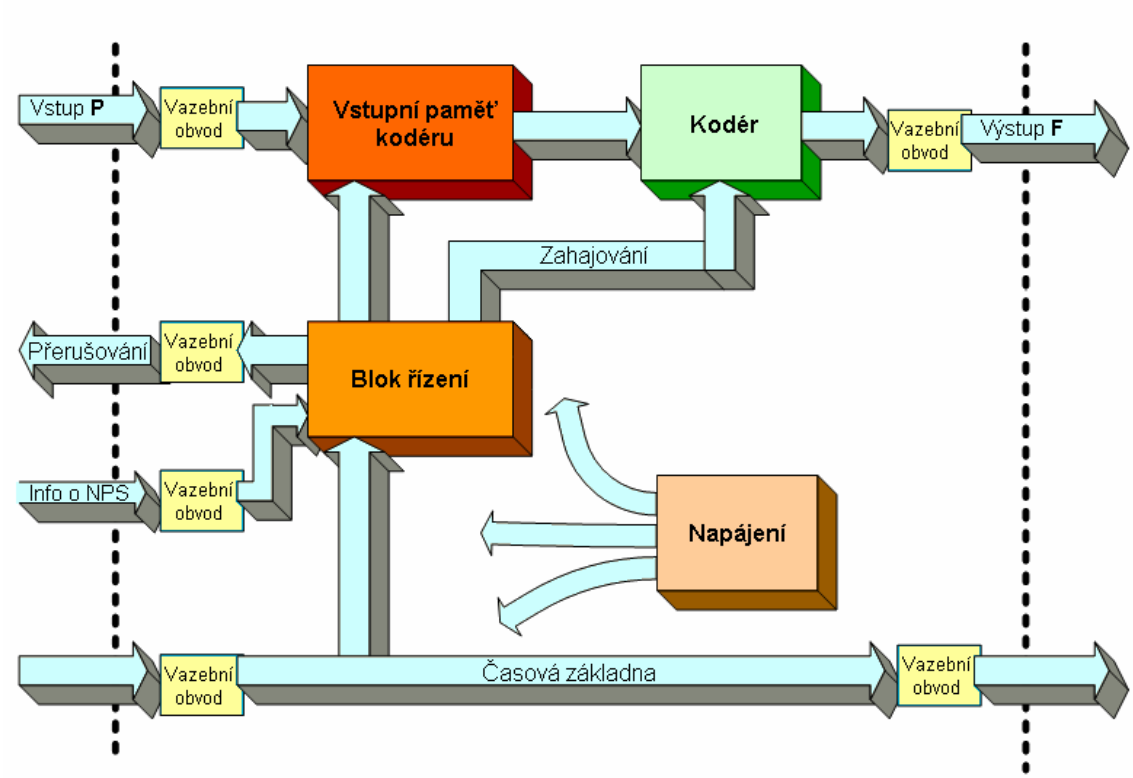
Tedy je třeba brát ohled i na ostatní zařízení, která budou společně s kodekem tvořit příslušný zabezpečovací systém daného protichybového kódového systému. Ve své podstatě zahrnuje protichybový kódový systém nejen samotné kódovací a dekódovací algoritmy (procesy), ale pracuje s dalšími zařízeními potřebnými pro realizaci kódu a mimo to obsahuje i řadu dalších zařízení zajišťujících správnou činnost těchto dílčích částí viz. [24].

PKS (protichybový kódový systém) je součástí NPS (nadřazeného přenosového systému). Umisťuje se (viz. předchozí kapitola) v blízkosti vstupních prvků na vysílací straně a výstupních prvků na přijímací straně nadřazeného přenosového systému. Toto začlenění je patrné z Obr. 6.2.



Obr. 6.2: PKS umístěný v NPS.

Vzhledem k poloze PKS v NPS je třeba respektovat řadu z toho vyplývajících požadavků. V rámci realizace kodeku je vhodné znát výskyt chyb a jejich rozložení v komunikačním kanále, vlastnosti a požadavky NPS (tyto požadavky zpravidla omezují výběrový prostor vhodných kódů) a také technické prostředky, jež lze pro realizaci použít. Obecně se kodér i dekodér protichybového kodeku skládá z několika částí: vstupní paměť, vlastní kodér (respektive dekodér), řízení kodéru, vazební obvody a napájení. Obecná struktura je uvedena na Obr. 6.3.



Obr. 6.3: Obecné blokové schéma korekčního protichybového kódového systému.

6.3 Synchronizace kodéru a dekodéru

Obecně na všechny PKS je kladen požadavek na alespoň dvě přenosové rychlosti a tedy i na různou synchronizační frekvenci určitých prvků. Konkrétní rychlost je na rozhraních kodéru či dekodéru a koncových zařízení komunikačních systémů pro něž je zprostředkováváno zabezpečení. Avšak jinou rychlostí pracuje rozhraní kodéru, dekodéru a přenosového kanálu, který je vzájemně spojuje.

Existují dvě základní možnosti, jak řešit problém s požadavkem na různé přenosové rychlosti [2]:

- Vstupní tok bitů protichybového kódového systému rozčlenit na přerušované úseky - pakety. Pak je možné, použít ve všech prvcích stejnou synchronizační frekvenci. Ovšem je třeba zajistit implementaci určitého komunikačního protokolu, jenž by řešil případné různé stavy zařízení - vysílající, připravené, vytížené, aj.
- V případě zachování nepřetržitého sériového toku informací se v systému vyskytnou minimálně dvě různé, výše popsané přenosové rychlosti. Jednotlivé prvky zabezpečovacího systému je třeba synchronizovat, respektive řídit odlišnými frekvencemi. Tato situace se řeší implementací speciálního bloku řízení viz. [2].

6.4 Způsoby návrhu a realizace číslicových systémů

Číslicový systém představuje zpravidla elektromagnetický systém, který pracuje s informacemi reprezentovanými v číselné formě - nejčastěji v binární soustavě a to za využití booleovy algebry. Typickým příkladem je běžný personální počítač, ovšem s číslicovými systémy se lze setkat i ve formě mikrokontrolerů a vestavěných systémů v běžně používaných spotřebičích a zařízeních - v automobilech, mobilních telefonech, mikrovlnných troubách, pračkách, aj.

V posledních desetiletích se dramaticky zvýšila využitelnost číslicových systémů v nejrůznějších oborech, a tudíž je snaha co nejvíce zefektivnit i zlevnit proces jejich návrhu a realizace [22]. Měnily a zdokonalovaly se postupy pro návrh a realizaci těchto systémů tak, aby bylo potřebné co nejméně úsilí, znalostí i zkušeností experta a naopak, aby byl proces návrhu co nejvíce automatizovaný a spoluřešený pomocí moderních návrhových systémů. Stále je ovšem možné vysledovat několik stupňů (popřípadě možností), kterými může proces návrhu číslicového systému projít. Obecně existuje řada způsobů návrhu číslicových systémů:

- Slovní popis - obsahuje veškeré podklady nezbytné pro přesnou definici činnosti obvodu a lze jej použít jako podklad pro fyzickou implementaci příslušného obvodu.
- Matematický popis - představuje formalizovaný slovní popis funkcí systému. Jeho největší význam spočívá ve verifikaci a validaci navrhovaného systému.
- Obvodové schéma - existují pokročilé návrhové prostředí podporující návrh číslicového systému pomocí zakreslování schématu. Velkou nevýhodou je ovšem snížená přehlednost u složitějších systémů a nutnost znalosti konkrétní představy o cílové architektuře navrhovaného systému.
- Programovací jazyk - v současnosti je nejvyužívanějším způsobem návrhu. Vytváří se popis chování obvodu ve vybraném programovacím jazyce pomocí vhodných jazykových konstrukcí - přiřazení, porovnání, smyčky, atd.

6.4.1 Digitální integrované prvky, obvody CMOS a TTL

Digitální integrované obvody obsahují logické členy a vykonávají určité pochody spojené se zpracováním digitálních signálů [38]. Základním stavebním blokem digitálních systémů je logický člen u kterého je jednoznačně dána jeho logická funkce. Stručně jsou uvedeny pouze obvody TTL a CMOS, existují ovšem i další jako například IIL nebo ECL, ale zejména zmiňovaný CMOS je v dnešní době perspektivní a také je nejrozšířenější.

Obvody TTL obsahují jako základní stavební jednotku logický člen NAND, který má několik vstupů a v pouzdře integrovaného obvodu je umístěno určité množství ekvivalentních logických členů. Každý typ obvodu od jiného výrobce může mít (a

zpravidla má) jiné vlastnosti. Například TTL řady 74 dosahují (dle výrobce) doby zpoždění logického členu při přechodu z úrovně L do úrovně H menší než 22 ns a při přechodu z úrovně H do úrovně L dokonce menší než 15 ns [38].

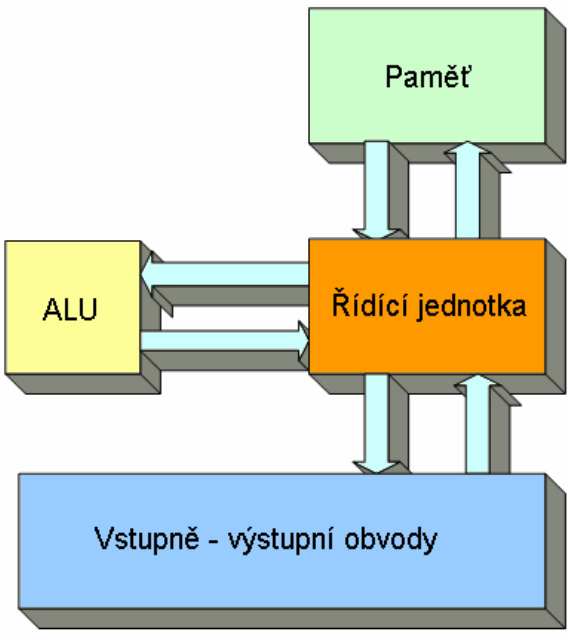
Obvody CMOS jsou perspektivní zejména z hlediska spotřeby, napájecího napětí a použitých technologií. Přednosti jsou ovšem vykoupeny menší proudovou vydatností výstupů a také proudovými špičkami, jenž vznikají při přepínání. Reprezentanty obvodů CMOS jsou řady 4000, 4500 a 14000. Běžně pracují uvedené integrované obvody s napájecím napětím od 3 do 15 V (mnohdy i do 18 V). Dle označení lze určit o jaký typ obvodu se jedná:

- AC (Advanced CMOS Logic) - vyznačují se nízkou spotřebou a výstupními proudy až 24 mA.
- HC (High-Speed CMOS) - jsou nejrozšířenější variantou. Dosahují relativně vysokých rychlostí, nízkého příkonu, vysoké šumové imunity a jsou finančně dostupné.
- AHC (Advance High-Speed CMOS) - obvody jsou obdobou HC, ale vykazují poloviční statický příkon.

Při realizaci daného protichybového kodeku by bylo nutné využít značného množství logických členů integrovaných v rámci pouzdra integrovaného obvodu.

6.4.2 Mikrokontrolery

Podstata a struktura mikrokontroleru je popsaná von Neumannovou architekturou, viz. Obr. 6.4. Například mikrokontroler MC68HC08 firmy Motorola má vnitřní generátor hodin s taktováním 13,8 MHz, 4 kB FLASH paměti, 128 B paměti RAM a pracuje v rozmezí 2,7 až 5,5 V.



Obr. 6.4: Von Neumanova koncepce.

Případné programování je téměř shodné se všemi ostatními mikrokontrolery dalších výrobců - odlišnosti mohou být v architektuře a v instrukcích jednotlivých typů. Programování probíhá v jazyku Assembler (Assembly Language), což je jazyk nejnižší úrovně. Tvoří jej symboly, které odpovídají zápisu strojového kódu. Příklad zápisu kódu v jazyce assembler je na Obr. 6.5. - patrné členění do čtyř skupin (sloupců). V prvním sloupci se uvádí návěstí řádku (nemusí se zapisovat), následuje instrukce (LDA je instrukce pro uložení určité hodnoty, zde uložení hodnoty bytu z adresy $30), ve třetím sloupci se uvádí parametry instrukce a na závěr je za znakem „ **;** “ popis, který nemá vliv na vykonání instrukce.

```
START    LDA      $30        ; NAČTENÍ DAT Z ADRESY 30
```

Obr. 6.5: Syntaxe psaní instrukcí v jazyce Assembler.

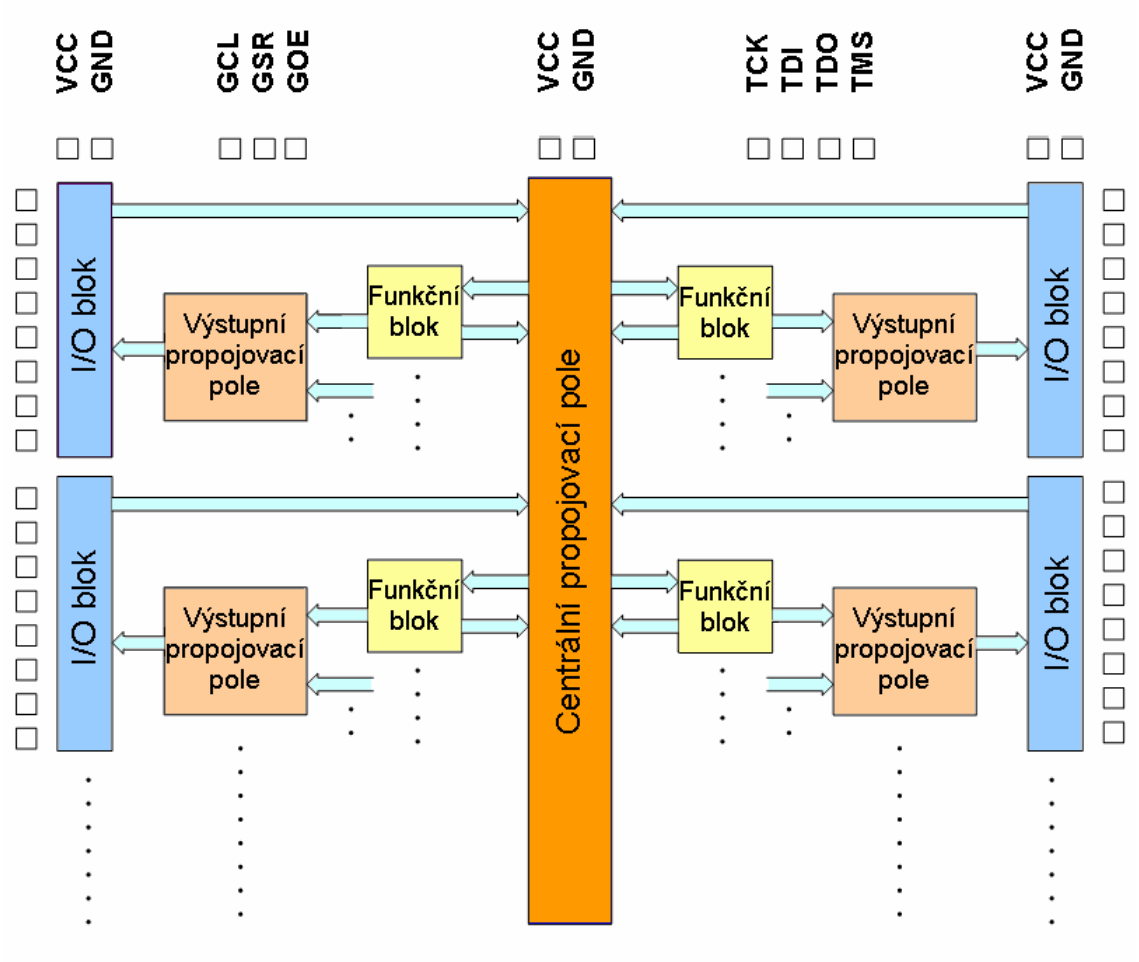
6.4.3 Programovatelné logické obvody

Z historického hlediska jsou relativně novým způsobem realizace digitálních obvodů programovatelné logické obvody (PLD – Programmable Logic Device). Jejich první generace je datována do let 1975 až 1983 (využívala se bipolární, TTL a CMOS technologie). Následovaly PLD typu PAL (II. generace 1983 až 1985), PLD typu GAL (III. generace 1985 až 1988) a dnes se nacházíme v době IV. generace programovatelných logických obvodů, které představují zejména PLD typu CPLD a FPGA [26].

CPLD - komplexní programovatelné logické obvody patří do skupiny elektricky reprogramovatelných PLD obvodů (EEPLD). Jejich struktura je tvořena programovatelnou maticí hradel AND následovanou hradlem OR a makrobuňkou. Požadovaná funkce na výstupu hradla OR je definována ve tvaru součtu součinů. Logické funkce se tvoří v makrobuňkách - nachází se ve funkčních blocích, viz. Obr. 6.6. Do AND matic každého z funkčních bloků lze zavést i několik desítek vstupních signálů a to v přímé i negované podobě. Na výstupu AND matice se nacházejí logické součiny. Dané součiny se pomocí bloku alokátoru součinů předávají mezi sousedními makrobuňkami a lze tak vytvářet i složité funkce sdružující několik makrobuněk.

Z Obr. 6.6 je patrná obecná bloková struktura CPLD viz. [26]. Pro tuto architekturu jsou typické čtyři základní bloky:

- velké centrální propojovací pole,
- programovatelné funkční bloky (AND matice a několik makrobuněk s alokátory součinů),
- výstupní propojovací pole,
- I/O bloky (vstupní/výstupní bloky).



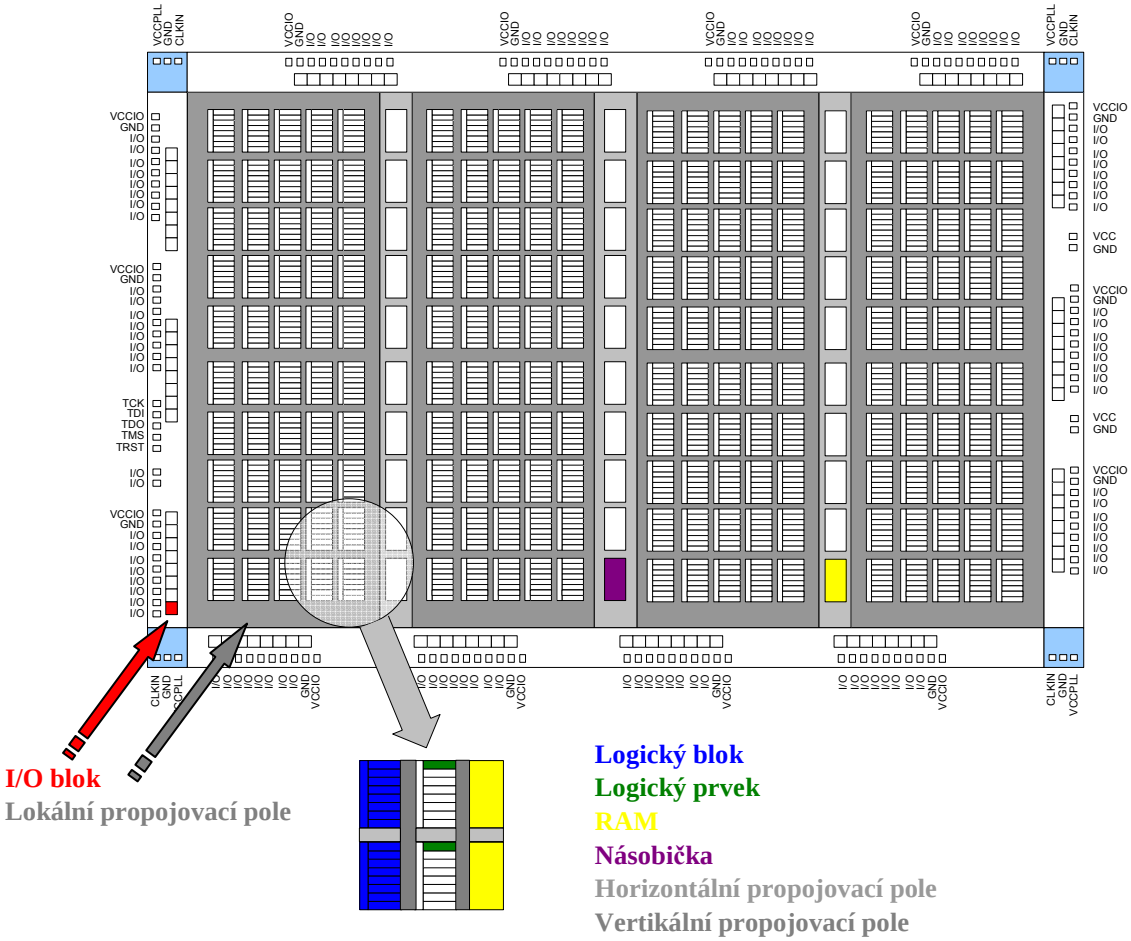
Obr. 6.6: Zjednodušené blokové schéma obecného CPLD obvodu.

Obvody umožňují realizovat složité kombinační logické, sekvenční logické nebo paměťové funkce, kde centrální propojovací pole plní funkci propojovacího systému všech vstupních a výstupních bloků a také všech makrobuněk (součásti programovatelných funkčních bloků).

Konfigurace jednotlivých prvků je udržována typicky v buňkách EEPROM (záleží ovšem na konkrétním řešení výrobce). EEPROM buňky umožňují přeprogramování obvodu v rozsahu několik tisíc cyklů [19]. Dnešní obvody CPLD dosahují velikostí typicky do 1024 makrobuněk a mohou obsahovat i 300 000 ekvivalentních hradel. Napájení obvodu je řešeno minimálně dvěma napájecími vývody. Prvním jsou napájeny vstupně-výstupní bloky, přičemž každá skupina bloků může být napájena napětím o jiné úrovni a druhé zajišťuje napájení jádra obvodu.

FPGA - programovatelná logická pole patří též do skupiny reprogramovatelných obvodů. Opět jsou tvořeny AND maticemi, OR hradly a makrobuňkami. Obvody FPGA jsou avšak složitější a zakládají se na malých generátorech logických funkcí s paměťmi, klopných obvodech a mnoha horizontálních i vertikálních propojeních. FPGA mají z programovatelných obvodů nejobecnější strukturu a obsahují nejvíce logiky.

Zjednodušené blokové schéma ukazuje Obr. 6.7. - jsou na něm vyznačeny programovatelné logické bloky, programovatelná horizontální i vertikální propojení a také programovatelné vstupní/výstupní bloky.



Obr. 6.7: Zjednodušené blokové schéma obecného FPGA obvodu.

Velkou odlišností od součástek CPLD je v jejich programování. Konfigurace FPGA je vždy při zapnutí napájení nahrávána z paměti (nejčastěji typu SRAM) a tedy funkce obvodu není okamžitě dostupná. Naproti tomu tato vlastnost umožňuje měnit konfiguraci za běhu a to i samotným hardwarem.

Tyto obvody mají několik napájecích vývodů - ze schématu, které je znázorněno na Obr. 6.7, jsou patrné například vývody VCC, VCCIO a VCCPLL. Jedno napájecí napětí je určeno pro jádro obvodu (VCC), další pro napájení I/O bloků (VCCIO) a také se zde nachází napájení fázových závěsů (VCCPLL) [40]. Mezi známé představitele obvodů FPGA patří řady Spartan a Virtex od firmy Xilinx.

6.5 Soubor kriterií pro výběr nejvhodnější varianty

Soubor základních kriterií pro výběr nejvhodnější metody pro realizaci kodeku v individuálních protichybových systémech obsahuje požadavky z několika různých oblastí. Jestliže jednotlivé požadavky nejsou považovány za stejně významné, je vhodné jim určit příslušný koeficient, jenž uvede daný požadavek na patřičnou hladinu vzhledem k ostatním zúčastněným hlediskům. Pro srovnání jednotlivých variant se zpravidla používá převod na normovanou veličinu, kdy danému hledisku příslušející nejlepší variantě se přidělí 100 bodů a ostatním variantám se přidělí poměrná část bodů, přičemž minimální normovanou hodnotou je 0 bodů [7], [24]. Výběr nejvhodnější varianty hardwarové implementace lze posuzovat dle následujících hledisek:

- cena realizace, finanční nároky,
- způsob popisu a realizace,
- nároky na napájecí napětí,
- možnost rozšiřitelnosti a modifikovatelnosti řešení,
- integrace řešení,
- náročnost realizace kodeku.

Cenové hledisko je samozřejmě vždy důležité, jelikož mnohdy rozhoduje o konkurenceschopnosti výrobků. Na implementaci v rámci použitého programovacího jazyka i jeho rozšířenost bere ohled způsob popisu a realizace. Případné nároky na napájení systému je nutné zohlednit. Ne vždy je možné nebo vhodné používat vyšší napájecí napětí. Další kritérium se snaží brát ohled na případné dodatečné rozšíření nebo změnu kodeku. Mohlo by se časem ukázat, že nadřazený přenosový systém vyžaduje větší či menší korekční schopnosti kodeku. Možnost obvod modifikovat ovšem ne vždy odpovídá vynaloženému úsilí a prostředky dosaženému cíli. Integrace zohledňuje řešení kodeku z pohledu počtu použitých součástek a jejich umístění v rámci pouzder integrovaných obvodů. Řešení s menším obsahem samostatně umístěných součástek má vliv na velikost výsledné desky plošných spojů, rozmístění součástek, složitost propojení atd. Pomocí kritéria náročnosti realizace lze posoudit úskalí použitého programovacího jazyka či hardwaru, jenž mohou částečně souviset i se zkušenostmi člověka v dané oblasti.

Na základě souboru kriterií lze učinit výběr vhodné metody pro hardwarovou implementaci, jenž může být znatelně ovlivněna přepočtem pomocí koeficientů, které se však mohou významně odlišovat u jednotlivých individuálních protichybových kódových systémů. Záleží především na které vlastnosti se klade hlavní důraz, jelikož mnohé požadavky vystupují proti sobě a je třeba hledat kompromis. Proto se při posuzování návrhu individuálních protichybových systémů často využívá hodnotící tabulky, v níž lze přehledně srovnat jednotlivé varianty.

6.6 VHDL

Vzhledem k uvedeným kritériím, perspektivnosti, využitelnosti se jeví nasazení FPGA obvodů pro individuální protichybové systémy jako správná volba [1], [7], [13]. Pro konstruktivní simulaci, syntézu komponent a realizaci představuje charakterizace pomocí některého jazyka pro popis hardwaru vhodný výběr. V současnosti je nejvyužívanějším způsobem návrhu vytváření popisu chování obvodu ve vybraném programovacím jazyce pomocí vhodných jazykových konstrukcí. Tyto jazyky se označují jako HDL (Hardware Description Language). Mezi nejznámější patří jazyk VHDL, jenž byl velmi brzy přijat i jako standart mezinárodní organizace pro elektrické a elektronické inženýrství, IEEE. Standardizování VHDL mimo jiné zapříčinilo rozšíření jazyka a jeho velkou přenositelnost, což představuje podstatnou výhodu při návrhu systémů vlastní realizace.

Při využití vypracovaných studií a simulací konvolučních kodérů a dekodérů pro digitální přenos signálů [48], [50], [56], se jako další vhodný krok nabízí pokračování v popisu pomocí jazyka VHDL, který je univerzální a přenositelný. Analýza na této úrovni představuje vlastní systémy realizace protichybového kódování, jelikož takto vytvořený systém je již možné použít a zařadit do nadřazených přenosových struktur. Tím bude dokončen rozbor rozšířeného popisu celého aparátu konvolučních kódů, jenž lze využít v individuálních protichybových systémech pro korekci shluků chyb.

První tři stupně popisu číslicového systému (slovní popis, matematický popis, obvodové schéma) byly používány při analýze i metodice návrhů individuálních protichybových systémů daných korekčních kódů [21]. Uplatněním čtvrtého stupně lze docílit realizace protichybového kódování [42]. Jelikož v dnešní době je možné hardware navrhovat technikami zažitými pro navrhování software i díky existenci technologie rekonfigurovatelného hardwaru. Což představuje další důležitý faktor, jenž zlevňuje i zefektivňuje proces návrhu a realizace číslicových systémů. Uvedené technologie umožňují jednoduše měnit funkci součástek určených jako cíl návrhu číslicového systému a tím usnadňují testování či úpravy na vyšší verzi. Souhrnně se všechny zmíněné součástky obvykle označují jako PLD. Realizaci systému lze provést na těchto dnes již dostupných výkonných procesorech, které navíc umožňují zpracování dat velmi vysokými rychlostmi [18], [32].

Základním používanými pojmy jazyka VHDL je komponenta a návrhová entita. Komponenta slouží pro práci s navrhovanými systémy a podsystémy bez nutnosti znalosti jejího popisu. K popisu komponenty slouží právě návrhová entita. Návrhová entita modeluje číslicový systém libovolné složitosti. Může jít o hradlo, klopný obvod, ale i počítačový systém. Každá návrhová entita je charakterizována svým rozhraním a jedním nebo několika alternativními těly. Rozhraní obsahuje definice společné všem tělům a zahrnuje především informace viditelné zvnějšku. Pro popis těla entity je možné využít dva základní styly popisu – behaviorální, strukturální.

Obecně poskytuje behaviorální popis nejvyšší míru abstrakce a tudíž i pohodlný styl programování a výsledná funkce navrhované entity je z jejího popisu snadno pochopitelná. Vysoká míra abstrakce (typická pro navrhování softwaru), ovšem nechává

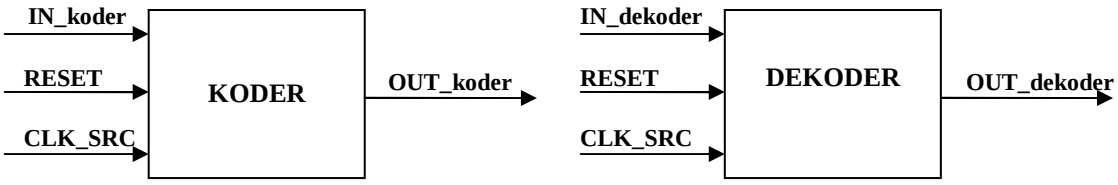
největší část práce na syntetizátoru výsledného hardwaru, který musí provést konkretizaci popisované entity a optimalizovat její strukturu. To ovšem nemusí dnešní syntetizační prostředky dokázat, a tudíž behaviorálně popsaná entita nemusí být ani syntetizovatelná - převeditelná do fyzické realizace. Uvedený styl popisu se především hodí využívat pro návrhy určené k simulaci.

Naproti tomu strukturální popis, je třeba tvořit s detailní znalostí vnitřní struktury výsledné entity a výsledná podoba popisu je pro neobeznámeného návrháře velmi obtížně čitelná. Avšak popis, detailně definující vnitřní strukturu entity dává jen velmi málo prostoru optimalizátoru a syntetizátoru pro rozsáhlejší zásahy do architektury výsledné komponenty. Pro kvalitní návrh určený pro fyzickou realizaci číslicového systému je vhodnější využít propracovaný strukturální popis entity, který bude bez problému syntetizovatelný a proces syntézy ovlivní podobu entity jen minimálně.

6.6.1 Univerzálnost komponent a rozhraní

Návrh řešení byl proveden v předchozích kapitolách odvozením jednotlivých kodeků a schémat jejich obvodového zapojení [44], [53]. V analyzované skupině konvolučních kódů nejméně logických operátorů k provedení příslušného zabezpečení potřebuje Iwadari-Masseyho kodek [56]. Na základě potřeby nejmenšího počtu logických operací viz. Tab. 4.1 představuje vhodnou jednoduchou volbu k vlastní implementaci komponent kodéru a dekodéru v programovacím jazyce VHDL. Jelikož u popisu pomocí VHDL může mít jedna entita více alternativních, různě popsaných, těl – provedou se oba způsoby popisu pro vyhodnocení přínosů a nedostatků.

Komponenty kodér a dekodér lze charakterizovat dvěmi různými popisy těla, přesto jejich základní rozhraní by mělo být jednotné. Konstrukt entity slouží k popisu rozhraní ve VHDL. Rozhraní entity obsahuje vstupní a výstupní porty. Vstupem a výstupem obou komponent je podle navržených obvodových schémat (viz. Obr. 4.3 a Obr. 4.4) sériový bitový tok určený ke kódování či dekódování. Dalším předpokladem je započetí přenosu z jistého definovaného vnitřního stavu komponent – vnitřní stav se nejlépe vyvolá signálem RESET. Vstupní sériový bitový tok je třeba vzorkovat, proto je nezbytné zajistit komponentám synchronizační signál, který se využije též v bloku řízení. Blokové schéma komponent kodéru a dekodéru naznačující pouze jejich základní rozhraní ukazuje Obr. 6.8.



Obr. 6.8: Blokové schéma komponenty kodéru a dekodéru.

Rozhraní entity ovšem také obsahuje takzvané generické parametry, pomocí kterých lze odlišit strukturu různých výskytů entity. Toto vystihuje jednotlivé obvodové návrhy kodéru a dekodéru, jejichž schéma zapojení je obecně dáno příslušnou blokovou vytvářecí maticí a zabezpečuje komunikaci proti zvolené délce shlukových chyb. Ve schématech byla použita proměnná určující vstupní dílčí tok k_0 a tato proměnná je i vhodným generickým parametrem, jelikož udává délku bloku informačních bitů zabezpečeného proti shlukové chybě. Změnou tohoto parametru se vygeneruje kodek, který je schopen zabezpečit přenášenou zprávu před shlukem chyb libovolně zvolené délky. Popis komponent s rozhraním uvedeným na schématu - Obr. 6.8 a s odvozeným generickým parametrem je v jazyce VHDL následující:

```
entity IwKoder is GENERIC (block_length: integer := 4);
    port (
        IN_koder  : in  std_logic;      -- Seriovy vstup koderu.
        OUT_koder : out std_logic;      -- Seriovy vystup koderu.
        RESET     : in  std_logic;      -- Reset (aktivni v 1).
        CLK_SRC   : in  std_logic       -- Synchronizacni signal, je
                                        --  entitou dale delen.
    );
end IwKoder;

entity IwDekoder is GENERIC (block_length: integer := 4);
     port (
        IN_dekoder  : in  std_logic;    -- Seriovy vstup dekoderu.
        OUT_dekoder : out std_logic;    -- Seriovy vystup dekoderu.
        RESET       : in  std_logic;    -- Reset (aktivni v 1).
        CLK_SRC     : in  std_logic     -- Synchronizacni signal, je
                                        --  entitou dale delen.
    );
end IwDekoder;
```

6.6.2 Popis strukturální

Teoreticky odvozené obvodové schéma je ideální pro přímý přepis do strukturálního kódu ve VHDL [49]. Před vlastní implementací je ovšem potřeba navrhnout řešení několika dílčích problémů: jak popsat opakující se strukturu danou dle vytvářecí blokové matice, jak přesně použít multiplexory a demultiplexory ve funkcích sériově-paralelních a paralelně-sériových převodníků a také jak řešit problém odlišnosti vstupní a výstupní rychlosti komponent.

Opakující se struktura se skládá z navzájem propojených registrů (paměťových buněk) a logických členů XOR, jejichž počet a velikost u registrů je odvoditelná z parametru k_0. U strukturálního popisu komponent lze využít pro popis opakování příkaz GENERATE, což je jistá forma makra. Příkaz GENERATE se často používá právě ve spojení s příkazy sloužícími k propojování jednotlivých komponent. Záhlaví příkazu GENERATE je analogické k záhlavím počítaných cyklů běžně používaných v softwarovém programování. K popisu určitých odlišností jednotlivých struktur generovaných pomocí uvedeného příkazu, se využije řídící proměnná tohoto počítaného cyklu. Pro vyjádření obecné struktury registrů počítajících zabezpečení v kodéru i v dekodéru, kde je třeba propojovat za sebe registry o určitých šířkách a jejich

jednotlivá propojení přivést na vstupy logických členů XOR, má příkaz v pseudokódu následující tvar:

```
FOR i IN pocatecni_hodnota TO konecna_hodnota GENERATE
 posuvny_registr(sirky f(i)):
                vstupem je vystup predchoziho registru
                vystup pripoj na XOR
                vystup pripoj na nasledujici registr
END GENERATE;
```

Pseudokód je třeba doplnit o konkrétní počáteční a koncovou hodnotu řídící proměnné, podobu funkce *f(i),* jež udává šířku registrů a v neposlední řadě o vlastní realizaci propojení prvků. První dva požadavky je možné odvodit z vytvářecí blokové matice (4.26) nebo schématu zapojení Obr. 4.3. Délka prvního registru je nepravidelná a délky ostatních registrů jsou střídavě konstantní (k_0+1) a střídavě členy aritmetické posloupnosti s koeficientem k_0 (tedy k_0, $2k_0$, $3k_0$ …) a s k_0 členy. Vzrůstající délku registrů je možné popsat právě pomocí řídící proměnné - její rozsah odráží počet členů zmíněné aritmetické řady. Dále je třeba v jednotlivých cyklech vystihnout střídání registrů konstantní délky, vzrůstající délky a nepravidelnost délky prvního registru. Upřesněný pseudokód by mohl vypadat následovně:

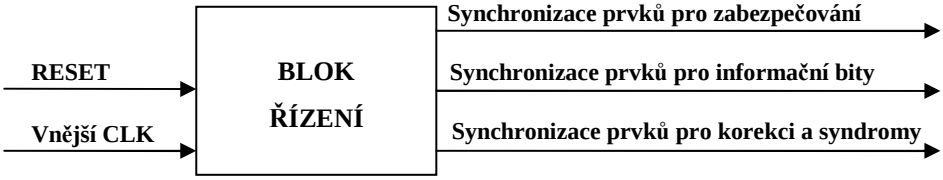
```
FOR i IN 1 TO k GENERATE
 Prvni cyklus:
       posuvny_registr(sirky n*k+1)
       posuvny_registr(sirky k)
       vzajemne propojeni registru a pripojeni na XOR
 Ostatni cykly:
       posuvny_registr(sirky k+1)
       posuvny_registr(sirky i*k)
       vzajemne propojeni registru a pripojeni na XOR
END GENERATE;
```

Kompletní provedení propojení, všech registrů, je podrobně zdokumentováno v přiložených zdrojových kódech.

Při implementaci je nutné vyřešit problém různého objemu dat na vstupu a výstupu kodéru i dekodéru. U každé z těchto komponent je buď vstup nebo výstup obohacen o zabezpečení, které oproti nezabezpečenému vstupu/výstupu zvětšuje počet zpracovávaných dat. V komponentách kodéru a dekodéru je potřeba různých synchronizačních signálů, jelikož jsou navrženy pro nepřerušovaný přenos dat. Potřebné jsou synchronizace pro práci se zabezpečeným tokem dat - to se týká prvků generujících výstup kodéru, zpracovávajících vstup dekodéru a s nezabezpečeným, čistě informačním, tokem dat - to se týká prvků zpracovávajících vstup kodéru či tvořících výstup dekodéru.

K těmto dvěma synchronizacím, jenž se obecně musí vyskytnout u všech protichybových kódových systémů pracujících s nepřerušovaným tokem dat, se navíc ještě využije synchronizace určitých prvků pro práci se syndromy a korekci bitů. Jelikož k přenosu zabezpečujícího bitu, výpočtu syndromu chyby i k případné korekci chyb v dekodéru dochází vždy po přenesení k_0 informačních bitů. Využijí se tři různé

synchronizační (řídící) signály. Komponenta blok řízení k výpočtu časování signálů používá externí hodinový signál a lze určit počáteční stav řízení - možnost resetovat komponentu. Tím je dáno veškeré rozhraní potřebné komponenty viz. Obr. 6.9.

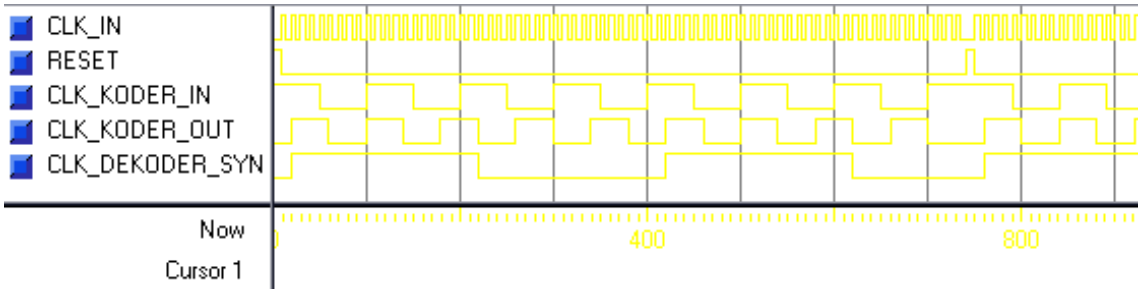


Obr. 6.9: Blokové schéma komponenty řízení.

Vzít v úvahu je nutné generičnost kodéru i dekodéru, jenž bude blok řízení řídit. S ohledem na generičnost pak tomuto rozhraní v popisu pomocí jazyka VHDL odpovídá následující konstrukce:

```
entity Control_block is GENERIC (block_length: integer := 4);
  port (
    RESET           : in std_logic;  -- Reset (aktivni v 1).
    CLK_IN          : in std_logic;  -- Synchronizace na vstupu.
    CLK_KODER_IN    : out std_logic; -- Synchronizace vstupnich dat
                                     --  vstupujicich do koderu.
    CLK_KODER_OUT   : out std_logic; -- Synchronizace zabezpecenych
                                     --  dat mezi koderem a
                                     --   dekoderem.
    CLK_DEKODER_SYN : out std_logic; -- Synchronizace
                                     --  zabezpecujicich bitu.
  );
end Control_block;
```

Jak vypadá příslušný časový průběh signálů komponenty pro délku bloku čtyř informačních bitů je uvedeno na Obr. 6.10. Bloky řízení podobného charakteru jsou tvořeny čítači realizující dělení vstupního synchronizačního signálu. Nejsnazší ovšem je popsat uvedenou komponentu behaviorálně za pomocí proměnných inkrementovaných podle hodnoty vstupní synchronizace. V tomto popisu syntetizátor rozpozná zamýšlené zapojení čítačů.



Obr. 6.10: Blok řízení - časový průběh signálů.

Pro zmíněnou komponentu je použita technika dělení hodinového signálu. Což způsobí, že komponenty kodeku budou pracovat na nižší frekvenci než je frekvence synchronizace dodaná bloku řízení. Tato technika je snadno použitelná, pokud cílová

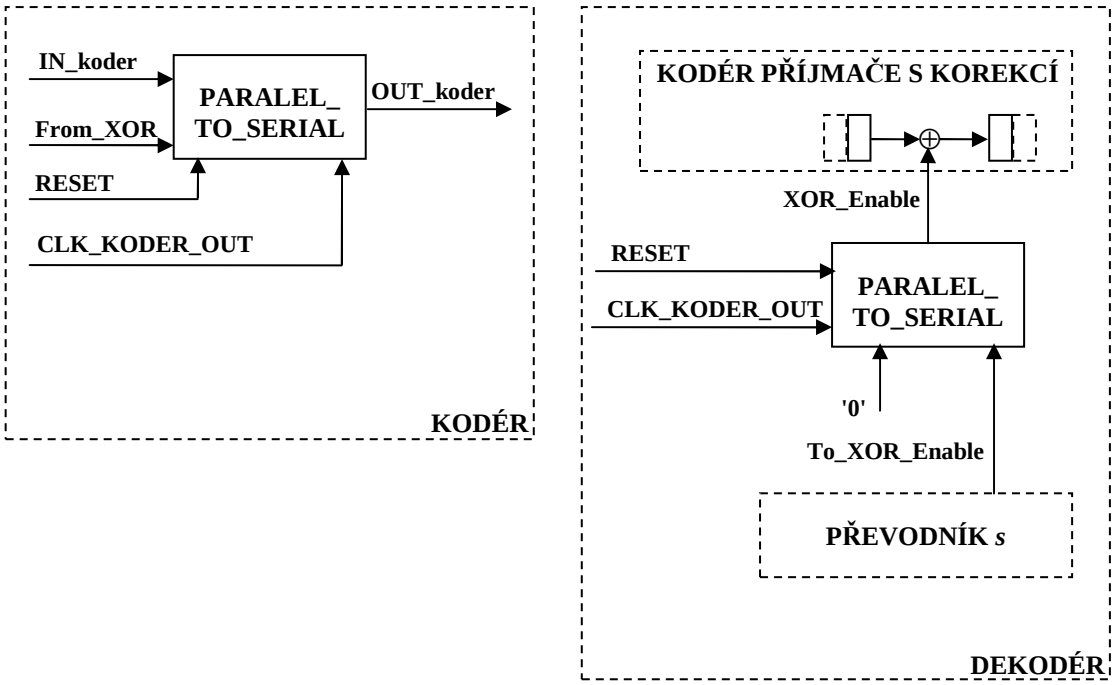
architektura disponuje dostatečně rychlou synchronizací na to, aby dělení neovlivnilo maximální pracovní frekvenci cílových obvodů. V opačném případě je možné použít bloky PLL určené pro násobení hodinového signálu, ovšem jedná se o náročnější techniku na realizaci a taktéž je méně přenositelná.

Přizpůsobení vstupu a výstupu kodéru i dekodéru je v obvodové realizaci navrženo použitím multiplexoru a demultiplexoru ve funkcích sériově-paralelního a paralelně-sériového převodníku [47]. Podrobnější analýzou je ovšem při realizaci možné najít efektivnější řešení. U vstupů obou komponent je dostačující, aby prvky pracující se vstupem jej vzorkovali s odpovídající frekvencí - navržená komponenta blok řízení. Pak zpožďovací posuvné registry kodéru načítají bity ze vstupu s frekvencí určenou signálem CLK_KODER_IN, se stejnou frekvencí načítají vstup zpožďovacích posuvných registrů dekodéru a s frekvencí CLK_DEKODER_SYN načítají vstup zpožďovací syndromové buňky dekodéru.

Výstup dekodéru je tvořen výstupem poslední paměťové buňky zpožďovacího posuvného registru viz. Obr. 4.4 a není třeba jej dále upravovat žádnou komponentou. Naproti tomu výstup kodéru viz. Obr. 4.3 je tvořen vstupními informačními bity a zabezpečujícími bity. Na výstup je potřebné přepínat bity informační i zabezpečující a to s takovou frekvencí, aby se v sériovém přenosu vyskytly všechny.

Komponenta musí tyto bitové toky přepínat na výstup s frekvencí určenou signálem CLK_KODER_OUT. Pokud komponenta bude mít dva vstupy, jeden pro informační bity, druhý pro zabezpečující bity a druhý vstup přepíná na výstup jednou po k_0 taktech, bude navržená komponenta využitelná kromě přepínaní výstupu kodéru také pro povolení opravy bitů v dekodéru. Prvky inicializující opravu pracují s frekvencí CLK_DEKODER_SYN, opravit je ovšem třeba pouze jeden bit v posuvném registru pracujícím na frekvenci CLK_KODER_IN - tyto frekvence také odpovídají frekvencím přepínaných vstupů v kodéru.

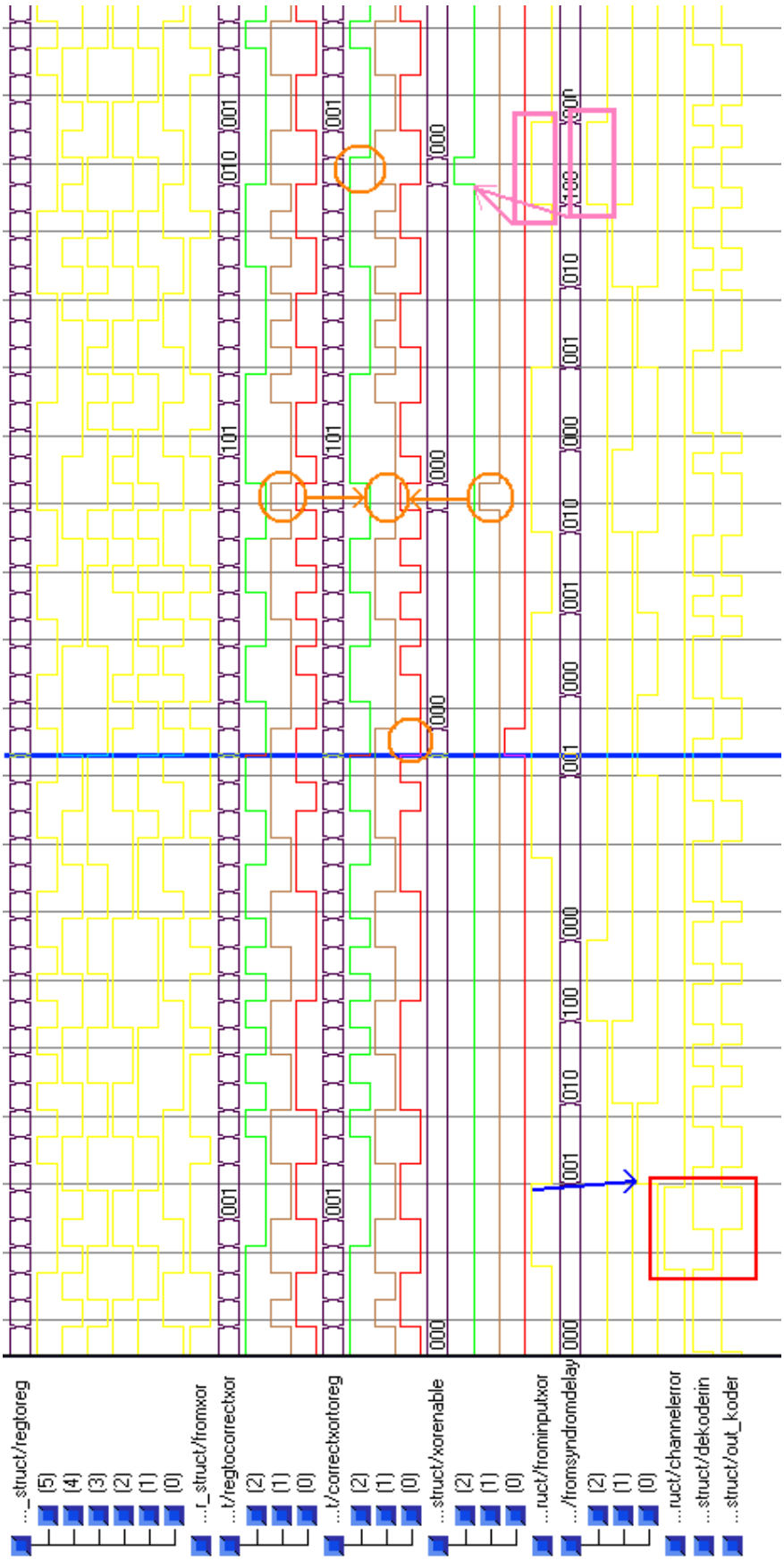
Povolení opravy vytvořené pomocí logického členu XOR je snadno realizovatelné přepínáním mezi signálem inicializujícím opravu a logickou hodnotou 0 - výsledkem výrazu s operaci XOR je vždy druhý operand a nedojde ke změně bitu. Blokové schéma popisované komponenty zapojené v kodéru pro přepínání výstupu a zapojené v dekodéru pro povolování oprav je naznačeno na Obr. 6.11. Komponentu lze popsat například pomocí čítače na základě jehož přetečení/nepřetečení se na výstup přepne první/druhý vstup. Vhodný je behaviorální popis za pomocí inkrementované proměnné, ve které syntetizátor snadno rozpozná zamýšlený čítač.



Obr. 6.11: Zapojení komponenty Paralel_To_Serial v kodeku.

V produktu Modelsim XE III 6.3c byla provedena simulace popsaných komponent za účelem verifikace modelovaných komponent - na odpovídající vstupy se dostanou očekávané výstupy. Ovšem též slouží i pro jejich validaci, především u cílových komponent kodéru a dekodéru - ověřování zda celá funkčnost modelu odpovídá původním neformálním popisům funkce. Na přiloženém CD má každá komponenta svoji testovací komponentu, která vytváří různé kombinace vstupů, pokud je to možné tak všechny kombinace. Na základě zobrazených výstupů je možné posoudit očekávanost chování - verifikovat model. Validací modelu se posuzuje zda signálové toky prokazují, že dochází ke korekcím poškozených bitů.

Tok vnitřní části signálů instance pro zabezpečování proti shluku čtyř chyb generické komponenty dekodéru podílejících se na detekci a korekci chyb představuje diagram na Obr. 6.12. Instance byla vytvořena nastavením generického parametru block_length na 3, což znamená že při přenosu mohou být poškozeny tři informační bity a jeden zabezpečující, aby byl dekodér schopen zprávu správně interpretovat. Vpravo dole, červeně orámovaná část, představuje průnik chyby do přenášené zprávy. Signál ChannelError představuje chybový vektor, který znehodnotil bity přenášené z kodéru signálem OUT_koder do vstupu dekodéru modelovaného signálem DekoderIN. Chyby v přenosu se postupně projeví v generátoru syndromu, který porovnává opětovně vypočtené zabezpečující bity s přenesenými. Zjištěné chyby se projeví jako impulzy signálu FromInputXor. Z generátoru syndromu postupují zjištěné příznaky chyb přenosu do zpožďovacích syndromových buněk - jeden případ znázorněn modrou šipkou.

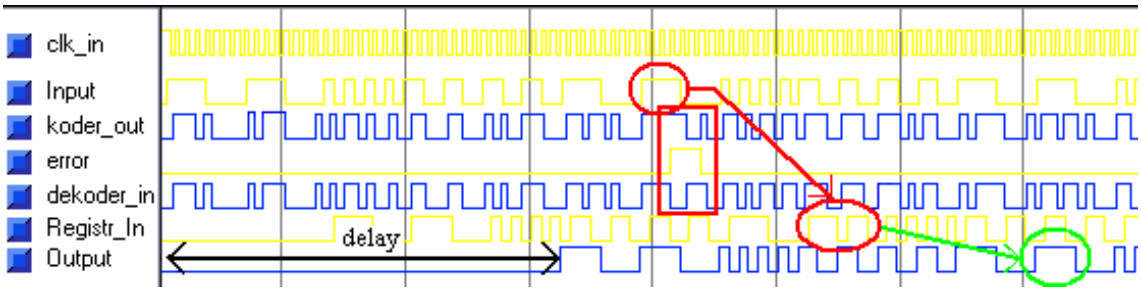


Obr. 6.12: Diagram signálových toků dekodéru při strukturálním popisu.

Zpožďovacími buňkami se pak bit příznaku chyby posouvá dokud generátor syndromu nevygeneruje další příznak, jenž způsobí iniciaci signálu XorEnable určeného k opravě chybného bitu a také vynulování zpožďovacích syndromových buněk - jeden případ iniciace vyznačen růžovou šipkou. Bit příznaku chyby se může ze zpožďovacích syndromových buněk vysunout ven aniž by inicioval opravu k tomu dojde v případě poškození zabezpečovacího bitu. K tomuto případu došlo v místě, kde je modrou šipkou zaznamenáno zpoždění vstupního syndromu - ten postupně projde všemi třemi buňkami bez iniciace signálu XorEnable. Signál XorEnable řídí vlastní opravu chybných bitů pomocí logické funkce XOR. Vektory signálů RegToCorrectXor a CorrectXorToReg představují bitový tok vstupující a vystupující do a z opravných logických členů XOR v posuvných registrech dekodéru. Na nich je viditelná oprava iniciovaná vektorem signálů XorEnable.

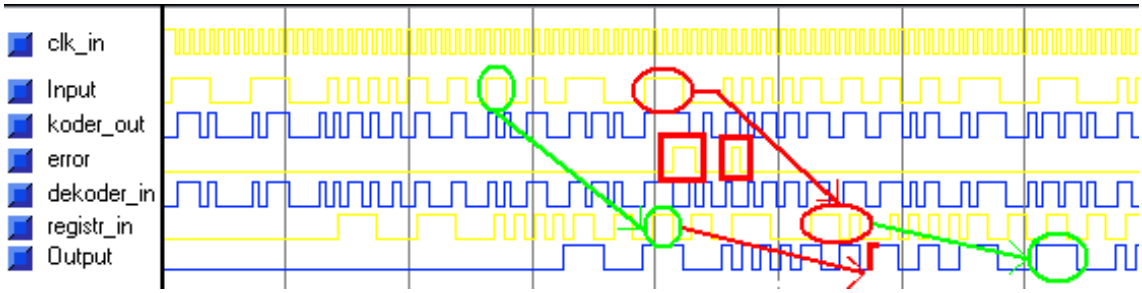
V Obr 6.12 je vždy stejnou barvou zakreslen signál vektoru RegToCorrectXor vstupující do opravného XORu, signál vektoru CorrectXorToReg vystupující z opravného XORu a signál vektoru XorEnable, který opravu řídí. Opravené bity jsou vyznačeny oranžovými kroužky, v jednom případě je oranžovým kroužkem označen i bit opravovaný, bit řídící opravu a šipkami je vyznačen vznik opraveného bitu.

Stejný diagram signálových průběhů zobrazený s větším rozlišením je pro názornost na Obr. 6.13. Znázorněn je vstup a výstup kodéru (Input, koder_out), chybový vektor (error) znehodnocující vstup dekodéru (dekoder_in), poškozený bitový informační tok vstupující do posuvného registru (Registr_in) a opravený zpožděný bitový informační tok vystupující z posuvného registru dekodéru (Output). V diagramu je červeným rámečkem znázorněn průnik chyby do přenosu. Červené ovály s šipkou znázorňují výskyt shlukové chyby délky čtyř bitů v informačních bitech, oprava shluku je znázorněna zelenou šipkou i oválem a černá šipka naznačuje informační zpoždění způsobené soustavou kodér-dekodér a odpovídá vztahu (4.41).



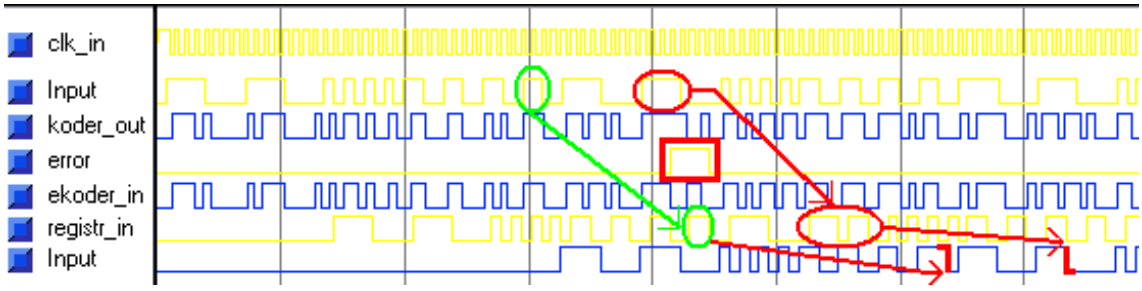
Obr. 6.13: Diagram signálových toků při korekci bitů.

Diagram signálových toků do a z komponent kodér a dekodér na obrázku Obr. 6.14 zobrazuje chování při nedodržení požadovaného ochranného intervalu *A*. V signálu error jsou vygenerovány chyby krátce po sobě. Dekodér poté sice nejen správně opravil vzniklou chybu v přenosu (znázorněno červenou šipkou od signálu Input k signálu Registr_in), ale též nesprávnou opravou poškodil jiný, správně přenesený bit (zelenou šipkou od signálu Input k Registr_in je naznačen bezchybný přenos, červenou šipkou k Output pak nesprávná oprava).



Obr. 6.14: Diagram signálových toků při nedodržení ochranného intervalu.

Demonstrace překročení zabezpečovací schopnosti kódu je na Obr 6.15. Se zachováním označování signálů, chybných a správných přenosů jako v předchozích diagramech. Projevuje se vliv příliš dlouhé chyby v přenosu. Opět dochází k pokusu o opravu na nesprávném místě a navíc k neúplné opravě přenosem poškozených bitů.



Obr. 6.15: Diagram signálových toků při nedodržení korekčních schopností kódu.

Naznačené varianty ukazující chování kodeku při nedodržení potřebných parametrů přenosu. Případy jsou si podobné, jelikož násobnou chybu lze v některých případech považovat za dlouhou shlukovou chybu a naopak. Dochází k chybným opravám správně přenesených bitů a k nedostatečným opravám poškozených bitů, avšak průnik chyby je vždy konečný.

6.6.3 Popis behaviorální

Používání speciálních bloků příkazů takzvaných procesů je typické pro behaviorální styl popisu. Pro správné rozdělení příkazů do procesů je třeba odlišit, v závislosti na kterých vstupech a signálech vnitřního stavu systému se mění výstupy a vnitřní stav popisovaného systému. Stejně jako u strukturálního popisu je třeba navrhnout řešení popisu opakujících se struktur a popis různých synchronizací pro vstupy a výstupy kodeku navíc i logické členy XOR a AND, posuvné registry, zpožďovací buňky a převodníky signálů o různé synchronizaci.

Pomocí stejnojmenných vestavěných logických operátorů jazyka VHDL je nejsnadnější namodelovat logické členy pro výpočet operací XOR a AND. Další potřebnou komponentou je posuvný registr, jehož behaviorální popis se sestává z vektoru signálů, měněných v těle procesu, který způsobí změnu stavu až při změně potřebného (synchronizačního) signálu. Činnost posuvu v registru lze popsat pomocí operátoru konkatenace (&) nebo logického posunu či rotace (SLL, SRL, SLA, SRA,

ROL a ROR) - více o použití operátorů viz. [19]. Následujícím kódem je pak možné popsat samotný registr:

```
signal BUFFER: std_logic_vector(X downto Y); -- Registr potrebne
                                             --  sirky.

process(RST, CLK)                        -- Zmena jen pri zmene
                                         --  CLK a RST.
begin
   if (RST='1') then                     -- Resetovani.
      BUFFER <= (others => '0');
   elsif (CLK'event) and (CLK='1') then  -- Pri nastupne hrane hodin.
                                         -- Pro posuv bud.
      BUFFER <= INPUT & BUFFER(7 downto 1);
                                         -- Anebo.
      -- BUFFER <= BUFFER SRL 1;
      -- BUFFER(BUFFER'length - 1) <= INPUT;
   end if;
end process;
```

Převodník signálů a vstupů/výstupů s různou synchronizací je poslední z používaných komponent. V behaviorálním popisu není třeba na výstup připojit nastálo výstup jedné komponenty, naopak lze v čase přepínat výstupy více komponent. Pro tento účel ve strukturálním popisu musela být vytvořena komponenta PARALEL_TO_SERIAL. V behaviorálním popisu se ovšem budou pouze pomocí správně řízeného procesu posílat na výstup kodéru střídavě informační a zabezpečující bity z různých částí kodéru.

Behaviorální styl popisu je charakteristický tím, že je v něm možné používat konstrukce typické pro jazyky, jenž popisují software - tedy rozhodovací bloky, cykly, atd. Opakující se struktura by v běžném programovacím jazyce vedla k popisu využívajícího počítaný cyklus. Zajištění zabezpečení by popisoval cyklus odpovídající tomuto pseudokódu:

```
-- Deklarace pomocnych promennych, pro moznost vyuziti cyklu.
pomocna_promenna;
pomocny_index;

-- Pred cyklem se popisuji nepravidelnosti.
pomocna_promenna := BUFFER(X) XOR BUFFER(Y)

-- Dalsi hodnoty jsou v registru rozmisteny pravidelne.
FOR i IN 2 TO k LOOP
  pomocny_index := pomocny_index + f(i);
  pomocna_promenna := pomocna_promenna XOR BUFFER(pomocny_index);
END LOOP;

-- Vysledne zabezpeceni se posle na vystup.
OUT <= pomocna_promenna;
```

Čítače při behaviorálním popisu lze využít i přímo v popisu funkce kodéru či dekodéru, kde na základě hodnoty těchto čítačů se propouští vstup do určitých prvků, popřípadě propouští výstup z určitých prvků na výstup celé komponenty. Nejdříve je

třeba odvodit způsob dělení vstupního hodinového signálu na signály o potřebných frekvencích. Potřebné jsou synchronizace pro vstup kodéru, výstup kodéru a výstup zabezpečujících bitů z kodéru - dekodér pak použije stejné synchronizace. Pokaždé, když do kodéru vstoupí postupně k_0 bitů, vystoupí z něj k_0 + 1 bitů - z toho je jeden zabezpečující. Synchronizační signály vstupu, výstupu a zabezpečujících bitů se musí tedy po zmíněných počtech period ocitnout ve výchozím stavu. Ze vstupního periodického synchronizačního signálu je nezbytné odvodit signály o následujících frekvencích.

Pro vstup kodéru:

$$f_{IN\_koder} = \frac{k_0}{t_B}. \tag{6.1}$$

Pro výstup kodéru:

$$f_{OUT\_koder} = \frac{k_0 + 1}{t_B}. \tag{6.2}$$

Pro prvky pracující se zabezpečením:

$$f_{SYN\_dekoder} = \frac{1}{t_B}, \tag{6.3}$$

kde t_B je doba potřebná pro zpracování jednoho bloku bitů nezabezpečeného vstupu do kodéru. Doba t_B tedy odpovídá k_0 periodám synchronizace vstupu, k_0 + 1 periodám synchronizace výstupu a jedné periodě synchronizace zabezpečení. Lze snadno odvodit, že nejmenší počet period vstupní synchronizace odpovídá nejmenšímu společnému násobků všech počtů period, které se do této doby vměstnají. Tedy počet period děleného signálu:

$$t_B = NSN(k_0; k_0 + 1; 1) * T_{divided}, \tag{6.4}$$

$$NSN(k_0; k_0 + 1; 1) = NSN(k_0; k_0 + 1) = k_0 * (k_0 + 1). \tag{6.5}$$

Ze vzorců (6.1) až (6.5) je již snadné odvodit periody potřebných signálu vyjádřené v počtech period dělené synchronizace.

Pro vstup kodéru:

$$T_{IN\_koder} = \frac{1}{f_{IN\_koder}} = \frac{t_B}{k_0} = \frac{k_0 * (k_0 + 1)}{k_0} * T_{divided} = (k_0 + 1) * T_{divided}. \tag{6.6}$$

Pro výstup kodéru:

$$T_{OUT\_koder} = \frac{1}{f_{OUT\_koder}} = \frac{t_B}{k_0 + 1} = \frac{k_0 * (k_0 + 1)}{k_0 + 1} * T_{divided} = k_0 * T_{divided} . \quad (6.7)$$

Pro prvky pracující se zabezpečením:

$$T_{SYN\_dekoder} = \frac{1}{f_{SYN\_dekoder}} = \frac{t_B}{1} = \frac{k_0 * (k_0 + 1)}{1} * T_{divided} ,$$
$$T_{SYN\_dekoder} = k_0 * (k_0 + 1) * T_{divided} . \quad (6.8)$$

Příslušné synchronizační (respektive řídící) signály se mění vždy po vypočteném počtu změn signálu dělené synchronizace. Vlastnímu dělení synchronizací pak může odpovídat diagram na Obr. 6.10 pro konkrétní hodnotu parametru k_0. V behaviorálním popisu komponent stačí použít inkrementované čítače, na základě jejichž hodnoty se vyvolá příslušná akce. Pseudokód řízení činnosti kodéru či dekodéru může vypadat následovně:

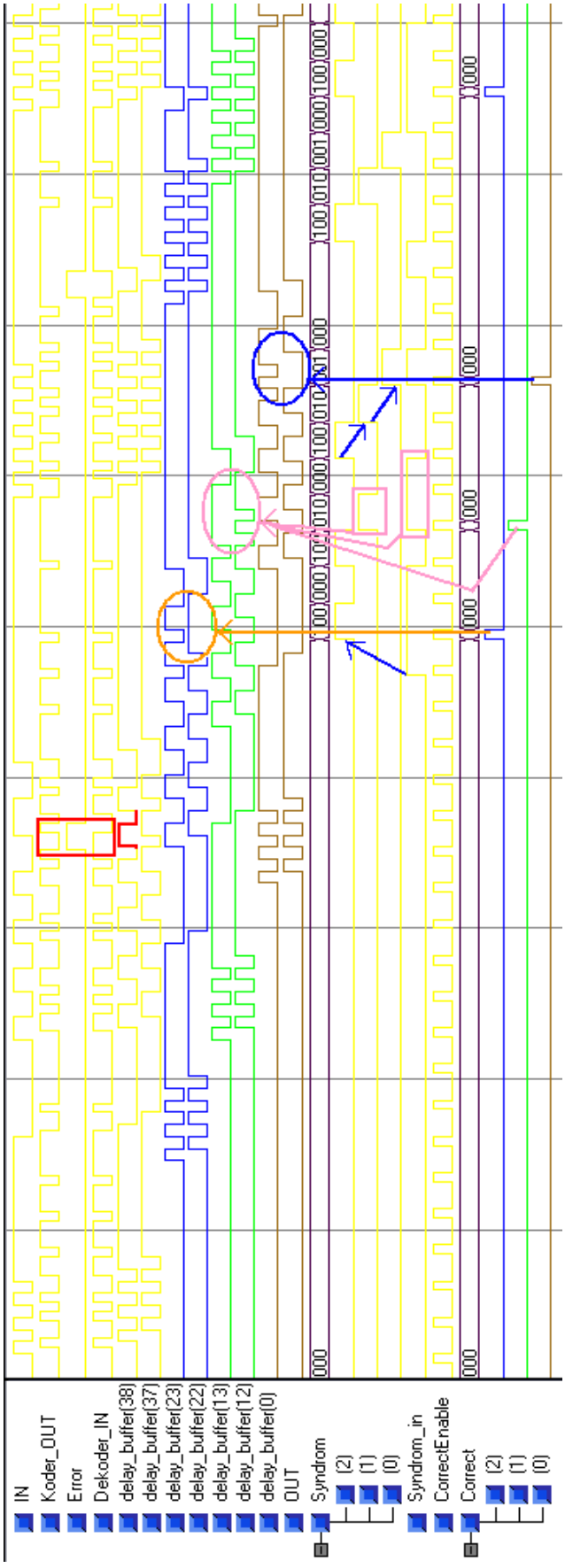
```
-- Deklarace promennych predstavujicich jednotlive citace.
counter_in;
counter_out;
counter_syn;

process (CLK, RESET)
begin
  if RESET = '1' then              -- Pri aktivite signalu reset.
     vynuluj vsechny citace;
  elsif CLK='1' and CLK'event then -- Pri nabezne hrane vstupnich
                                   --  hodin pokud citac dosahl
      if (counter_in = k+1)        --   hranicni hodnotu
         vynuluj counter_in;       --    nuluj
         nacitej vstup;            --     a vykonej prislusne akce.
      else
         inkrementuj counter_in;   -- Jinak citac jen inkrementuj.
      endif;

      if (counter_out = k)         -- Obdobne pro všechny citace.
      ...
      ...
end process;
```

Hodnoty čítačů řídí funkci celého kodéru i dekodéru. Z uvedeného příkladu je zřejmé vyčítání vstupních bitů ze sériového proudu o předpokládané frekvenci. Stejně ovšem budou pomocí všech čítačů řízené i ostatní činnosti, jako například posílání informačních a zabezpečujících bitů na výstup nebo oprava chybných bitů v dekodéru. Takto je provedena plná náhrada funkce prvku PARALEL_TO_SERIAL, který zmíněné činnosti řídil v kodéru či dekodéru popsaném strukturálně. Pomocí celočíselné proměnné (typu integer) je možné jednoduše modelovat vlastní čítač, avšak po syntéze bude mít čítač vždy pevnou šířku 32 bitů. Lepší variantu představuje popsat čítač pomocí bitového vektoru o šířce odpovídající nejbližší vyšší celočíselné hodnotě

dvojkového logaritmu požadované maximální hodnoty čítače. V tomto případě se dosáhne stejného výsledného efektu s menší spotřebovanou plochou výsledné obvodové realizace.

Analogické je testování behaviorálně popsaných komponent s výše uvedeným postupem jako u strukturálně popsaných komponent. Tok vnitřní části signálů instance pro zabezpečení proti shluku čtyř chyb generické komponenty dekodéru podílejících se na detekci i korekci chyb představuje diagram na Obr. 6.16. Mechanizmus vzájemné inicializace jednotlivých signálů a následná korekce chyb je analogická jako na Obr. 6.12. Zde velmi stručný popis se zaměří převážně na odlišnosti oproti situaci se strukturálně popsaným systémem. V diagramu je červeně vyznačen průnik chyby do přenosu, jenž v dekodéru v jistých okamžicích vyvolá vznik vstupních syndromů (Syndrom_in), které se zachytávají v syndromových buňkách (Syndrom). Syndromové buňky spolu se vstupním syndromem vyvolávají opravu daných paměťových buněk (signály Correct).

Příklady demonstrující korekční schopnosti modelovaného protichybového kódového systému, které by zobrazovaly pouze vstup a výstup systému a průnik chyby, jsou naprosto identické s příklady uvedenými v kapitole 6.6.2. Behaviorálně a strukturálně popsané komponenty se liší pouze vnitřními signály a aktuálním vnitřním stavem signálů, na totožné vstupy však reagují totožnými výstupy, mají tedy ekvivalentní chování [26].



Obr. 6.16: Diagram signálových toků dekodéru při behaviorálním popisu.

6.6.4 Parametry a syntéza komponent, konečná obvodová podoba

Pomocí vývojového prostředí Xilinx ISE 10.1 byly syntetizovány zdrojové kódy popisující kodek. Pro porovnání výsledných komponent popsaných strukturálně a behaviorálně byly využity charakteristické parametry a výsledné obvodové zapojení komponent vygenerované v průběhu syntézy.

Mezi nejvýznamnější kritéria pro hodnocení a porovnávání číslicových systému patří jejich rychlost a plocha. Pro hodnocení behaviorálně a strukturálně popsaného kodeku byly rovněž použity tyto kritéria a to konkrétně maximální dosažitelné pracovní frekvence a rozsah obvodu přepočítaný do takzvaných ekvivalentních hradel. Byla použita struktura FPGA s čipem Spartan-3 do níž je a aplikován Iwadari-Masseyho kodek. Parametry pro zvolené hodnoty opravitelné délky shluku chyb b = 4, 5, 9 a 10 jsou znázorněny v Tab. 6.1 a 6.2.

Tab. 6.1: Hodnoty maximální pracovní frekvence kodeku.

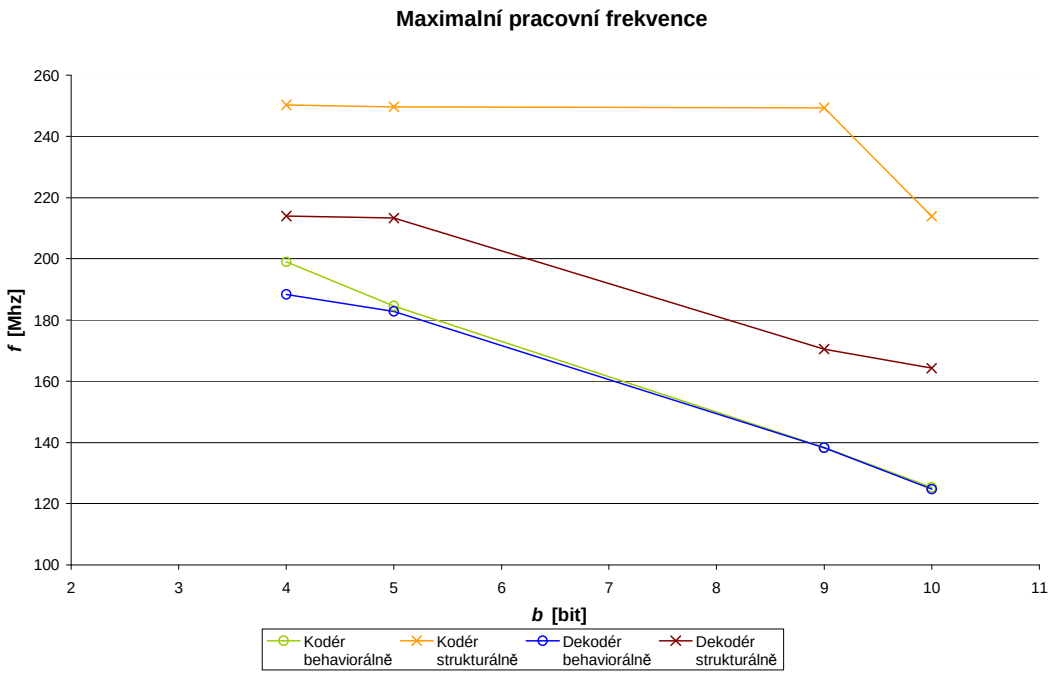
		FREKVENCE [MHz]			
		kodér behaviorálně	kodér strukturálně	dekodér behaviorálně	dekodér strukturálně
b [bit]	4	198,958	250,251	188,395	213,995
	5	184,575	249,626	182,816	213,311
	9	138,358	249,378	138,282	170,533
	10	125,293	213,904	124,701	164,285

Tab. 6.2: Hodnoty objemu použité logiky v ekvivalentních hradlech.

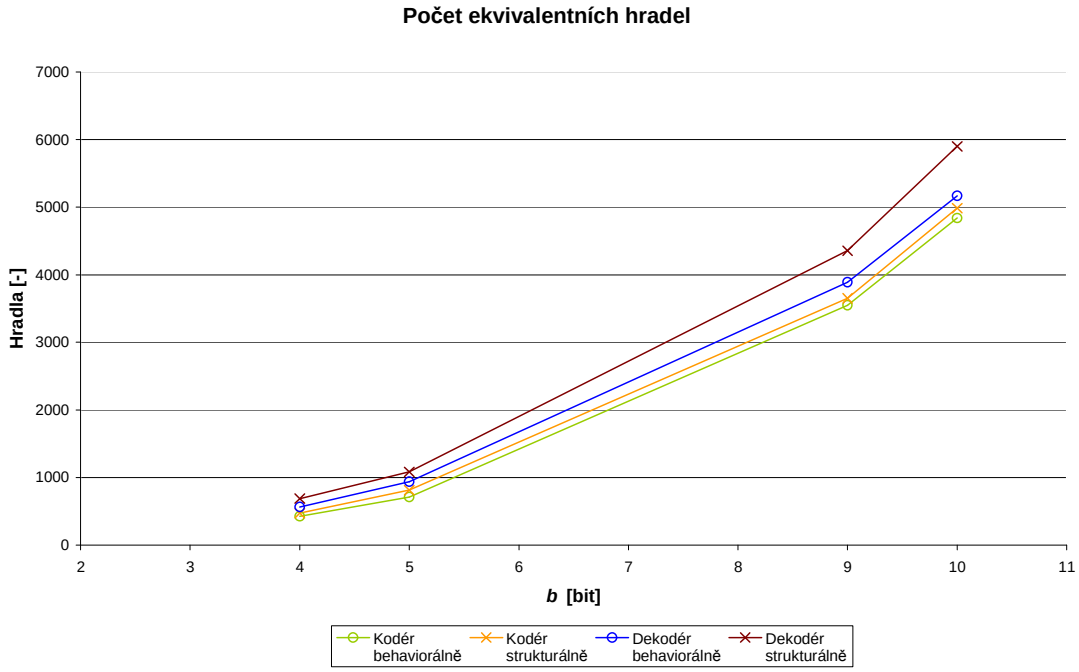
		EKVIVALENTNÍCH HRADEL			
		kodér behaviorálně	kodér strukturálně	dekodér behaviorálně	dekodér strukturálně
b [bit]	4	427	475	565	691
	5	713	814	940	1083
	9	3549	3652	3891	4356
	10	4838	4985	5168	5899

Z hodnot Tab. 6.2 a především z Obr. 6.18 je zřejmé, že plocha na čipu spotřebovaná logikou tvořící kodér či dekodér Iwadari-Masseyho kódu se zvyšující se hodnotou korekční schopností prudce roste, což je v souladu s teoreticky vyjádřenými vztahy pro počet paměťových buněk a logických operátorů. Objem spotřebované logiky není příliš závislý na tom zda je komponentou kodér, či dekodér a zda byla syntetizována z behaviorálního či strukturálního popisu i když o něco méně logiky spotřebovává kodér než dekodér a nepatrně lepší vlastnost v tomto ohledu mají behaviorálně popsané komponenty. V Tab. 6.1 a Obr. 6.17 je zaznamenána závislost maximální dosažitelné pracovní frekvence komponent v závislosti na hodnotě

generického parametru. Je zřejmé, že hodnoty pracovní frekvence klesají se zvyšující se korekční schopností a že strukturálně popsané komponenty v tomto ohledu poskytují výrazně lepší vlastnosti.

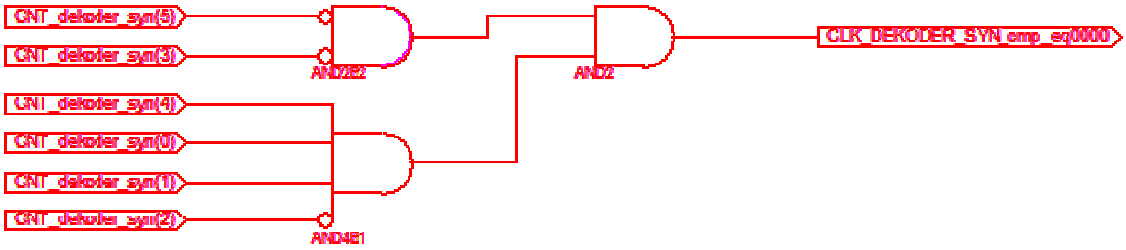


Obr. 6.17: Závislost maximální pracovní frekvence kodeku.

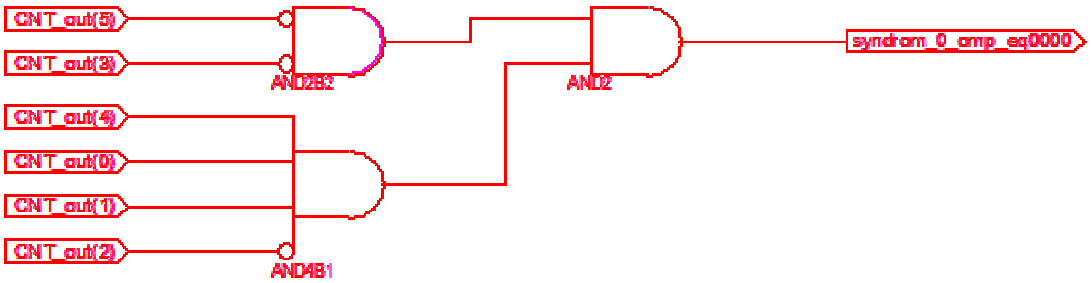


Obr. 6.18: Závislost objemu použité logiky kodeku.

Rozsah i způsob popisů cílových komponent v behaviorálním a strukturálním stylu se velice liší přestože, vycházely oba popisy ze stejných předloh viz. Obr 4.3 a Obr. 4.4. Avšak ekvivalence činnosti komponent byla ověřena a potvrzena v předchozích částech. Bylo ukázáno, že i charakteristické parametry komponent popsané v těchto dvou stylech jsou velmi blízké. Definitivní posouzení odlišností poskytuje porovnání RTL schémat vygenerovaných při syntéze - obvykle soubory s příponou .ngr. Vygenerovaná schémata se zdají být po zběžném prohlédnutí značně odlišná, což je ovšem způsobeno odlišným členěním prvků do hierarchických bloků. Po podrobném prozkoumání schémat je zřejmé, že jsou si výsledné obvody velmi podobné, v jistých částech jsou často naprosto identické - například obvody synchronizace viz. Obr. 6.19 a Obr. 6.20.

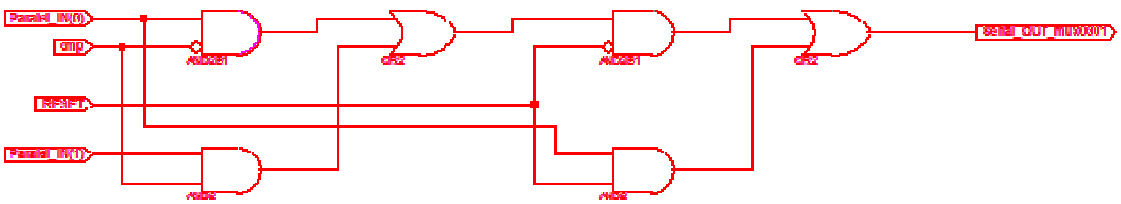


Obr. 6.19: RTL schéma obvodu synchronizace - strukturálním popis.



Obr. 6.20: RTL schéma obvodu synchronizace - behaviorální popis.

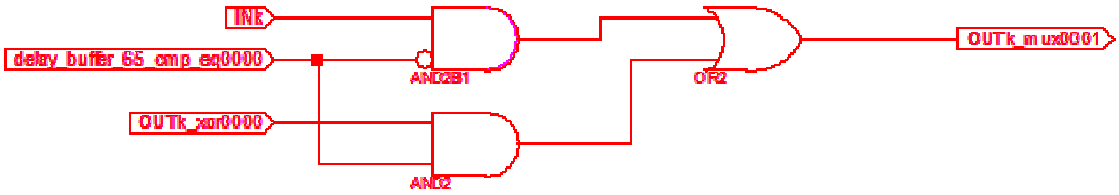
Značně odlišně byl řešen v obou popisech problém související s přepínáním informačních a zabezpečujících bitů na výstup dekodéru. Obvodová schémata obvodů realizujících vlastní přepínání jsou zobrazena na Obr. 6.21 a Obr. 6.22. K obdobným schématům opět vedly různé popisy, kdy odlišnosti lze nalézt u řešení RESETu.



Obr. 6.21: RTL schéma obvodu přepínajícího výstup kodéru - strukturální popis.

Pravděpodobně jediným místem, kde se výsledná schémata popsaná v různých stylech liší jsou obvody realizující vlastní korekci bitů. Ze schématu pro behaviorálně

popsaný dekodér je jen velmi těžko rozpoznatelné, které obvody se přesně na této činnosti podílejí. Oproti dobře čitelnému a logicky strukturovanému schématu pro strukturálně popsaný dekodér.



Obr. 6.22: RTL schéma obvodu přepínajícího výstup kodéru - behaviorální popis.

7 Závěr

Zabezpečovací kódování patří k nepostradatelné části téměř každého přenosového systému. Jelikož při přenosu informace kanálem dochází k jejímu poškození kvůli různorodým vlivům šumu. Problémem shluků chyb se v současných systémech musí zabývat stále více zařízení. Hlavní příčinu představuje rychlý rozmach vysokorychlostních systémů pro výměnu, zpracování i uchování dat. K potlačení shluků chyb či k opravě chybných bitů se využívají rozličné metody, jejichž stručná charakteristika i principy byly představeny v úvodu.

Pro rozšíření vhodných použitelných systémů pro uplatnění v oblasti korekce shlukových chyb je využito alternativního řešení korekce shlukových chyb pomocí vybraných systematických konvolučních kódů. Jelikož v individuálních protichybových systémech je vhodné zvážit použití jiných zabezpečovacích kódových technik než masová aplikace stávajících robustních řešení pro dosažení maximální efektivity, které lze dosáhnout v některých případech i s jednoduššími systémy. Proto pro soubor systematických konvolučních kódů je proveden rozbor i srovnání jejich zabezpečovacích schopností.

Účelem je poskytnout adekvátní náhradu hromadně používaných systémů pro opravu shluků chyb, jejichž nasazení v individuálních protichybových systémech umožní zlepšení přenosu dat. Odvozen je podrobný matematický aparát pro rozšíření souboru kritérií, která umožní vhodnější srovnání protichybových systémů pro přenos digitálních signálů. Získané vztahy s využitím blokové vytvářecí matice mají všeobecnou platnost. Uplatnění naleznou při návrzích individuálních protichybových systémů, kdy bez předchozí podrobné znalosti daného systému se získá více popisných informací.

Na základě konfrontace s jinými nejčastěji aplikovanými způsoby ochrany zprávy před shlukem chyb se potvrdila existence mezery používaných systémů pro korekci shluků chyb různé délky mezi systémy používající základní blokové kódy na straně jedné a sofistikovanými systémy na straně druhé. Dle získaných výsledků lze v individuálních protichybových systémech dosáhnout lepších vlastností využitím skupiny konvolučních kódů. Kvůli prověření jejich správnosti jsou teoretické výsledky podrobeny kontrole pomocí simulace.

Vizualizace a simulace implementování algoritmů zabezpečovacích kódů v grafickém uživatelském rozhraní Matlab je provedena za účelem dostatečně názorného přiblížení jednotlivých fází kódovacích i dekódovacích postupů. Obdržené výsledky simulace potvrzují správnou funkci příslušného kodeku. Tedy v kapitole 4.2, 4.3, 4.4 odvozená rozšiřující kritéria popisu kódů byla realizovanou simulací dostatečně prověřena. Veškeré takto získané výsledky jsou hojně publikovány.

Na závěr je ověřena možnost realizace individuálních protichybových systémů, kdy je třeba uvažovat součinnost zařízení nejen pro bezprostřední realizaci kódu. I při

výběru vhodné metody pro hardwarovou implementaci záleží na vlastnostech, na které se klade hlavní důraz, jelikož mnohé požadavky vystupují proti sobě a je třeba hledat kompromis. Vzhledem k další využitelnosti získaných výsledků, zefektivňování procesu návrhu i realizaci číslicových systémů a jejich snadné přenositelnosti je možný způsob realizace demonstrován pomocí programově logických obvodů, protože i díky rekonfigurovatelnosti je v dnešní době možné hardware navrhovat i technikami dříve určenými jen pro návrh softwaru.

Analýza struktury pomocí popisu VHDL představuje vlastní systémy realizace protichybového kódování. Takto vytvořený systém je již možné použít a zařadit do nadřazených přenosových struktur. Opět se potvrdily teoreticky získané výsledky a též se potvrdila ekvivalence činnosti komponent v behaviorálním i strukturálním stylu, přestože jejich popis se značně liší. Navržený rozšířený matematický popis systematických konvolučních kódů i možnost jejich realizace v individuálních protichybových systémech lze pokládat za dostatečně prověřený. Všechny nově získané poznatky z této práce, lze uplatnit u návrhů individuálních protichybových systémů zaměřených na korekci shluků chyb.

8 Literatura

[1] BASALAMAH, A., SATO, T. A Comparison of Packet-Level and Byte-Level Reliable FEC Multicast Protocols for WLANs. *In Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, 2007, pp. 4702 - 4707, ISBN 978-1-4244-1043-9.

[2] BLAUM, M. Enhanced decoding of interleaved error correcting codes. *IEEE International Symposium on Information Theory*, 1995, pp. 412 – 414, ISBN 0-7803-2453-6.

[3] BRUEN, A., FORCINITO, A. *Cryptography, information theory, and error-correction: A handbook for the 21st century*. Hoboken, N.J.: Wiley-Interscience, 2005, 468 p., ISBN: 0-471-65317-9

[4] CELANDRONI, N. Comparison of FEC types with regard to the efficiency of TCP connections over AWGN satellite channels. *In Wireless Communications, IEEE Transactions*, 2006, pp. 1735 - 1745, ISSN 1536-1276.

[5] CLARK, C., CAIN, B. *Error-Correction Coding for digital communications*. New York: Plenum Pr., 1982, 436 p., ISBN 0306406152.

[6] EROZAN, K., BANE, V. *Advanced error control techniques for data storage systems*. Boca Raton, FL: CRC Taylor & Francis, 2006, 288 p., ISBN: 0-8493-9547-X.

[7] FARRUGIA, R., A., DEBONO, C., J. A Statistical Bit Error Generator for Emulation of Complex Forward Error Correction Schemes. *In Communications, 2007. ICC '07. IEEE International Conference*, 2007, Glasgow, pp. 177 - 182, ISBN 1-4244-0353-7.

[8] FENG, W., YUAN, J. A code-matched interleaver design for turbo codes. *Communications, IEEE Transactions*, 2002, Vol. 50, Issue 6, pp. 926 – 937, ISSN 0090-6778.

[9] GOFF, S., KHOO, K., TSIMENIDIS, C. Constellation Shaping for Bandwidth-Efficient Turbo-Coded Modulation With Iterative Receiver. *IEEE Transactions on Wireless Communications*, 2007, Vol. 6, pp. 2223 – 2233, ISSN 1558-2248.

[10] HAMOUDA, A., MCLANE, J. Space-time MMSE multiuser detection in multipath channels with RS coding. *IEEE International Conference on Communications*, 2001, pp. 1177 – 1181, ISBN 0-7803-7097-1.

[11] HANUS, S. *Bezdrátové a mobilní komunikace.* Skriptum FEKT VUT Brno, RadioMobil, a.s., Praha 2003, 134 s., ISBN 80-214-1833-8.

[12] HEEGARD, CH., WICKER, S. *Turbo Coding*. 3 edition, Springer; 2000, 240 p., ISBN-0-7923-8378-8.

[13] HELLGE, C., SCHJERL, T., WIEGAND, T. Multidimensional Layered Forward Error Correction Using Rateless Codes. *In Communications, 2008. ICC '08. IEEE International Conference*, 2008, Beijing, pp. 480 - 484, ISBN 978-1-4244-2075-9.

[14] JAIKWAN, J. *The Simulation, Modeling and Analysis of Wireless Local Area Networks Supporting the IEEE 802.11 Standard*. Storming Media, 1998, 101 p., ISBN 1423555031.

[15] JEONGSEOK, H., MCLAUGHLIN, W. Low-density parity-check codes over Gaussian channels with erasures. *Information Theory, IEEE Transactions*, 2003, Vol 49, Issue 7, pp. 1801 – 1809, ISSN 0018-9448.

[16] JURCA, D., FROSSARD, P. Optimal FEC rate for media streaming in active network. *In Multimedia and Expo, 2004. ICME '04. IEEE International Conference*, 2004, Taipei, Vol. 2, pp. 1319 - 1322, ISBN 0-7803-8603-5.

[17] KARBAN, P. *Výpočty a simulace v programu Matlab a Simulink*. Computer Press, Brno 2006, 220s., ISBN 80-251-1301-9.

[18] KIVIOJA, M., ISOAHO, J. Design and Implementation of Viterbi Decoder with FPGAs. *The Journal of VLSI Signal Processing*, 1999, Vol. 21, Issue 1, pp. 5 - 14, ISSN 0922-5773.

[19] KOLOUCH, J. *Programovatelné logické obvody a modelování číslicových systémů v jazycích ABEL a VHDL*. Skriptum VUT Brno, Brno 2000, 83 s., ISBN 80-214-1733-1.

[20] LIU, H. et al. Staggered FEC System for Seamless Handoff in Wireless LANs: Implementation Experience and Experimental Study. *In Multimedia, 2007. ISM 2007. Ninth IEEE International Symposium*, 2007, Taichung, pp. 283 - 290, ISBN 978-0-7695-3058-1.

[21] MORELOS-ZARAGOZA, R. *The Art of Error Correcting Coding*. 2nd ed., John Wiley & Sons Ltd., 2006, 384 p., ISBN 0-470-44782-4.

[22] MUKARAMI, K. *VoIP Evaluation for MBWA*. Project IEEE 802.20 Working Group on Mobile Broadband Wireless Access, 2005, 11 p.

[23] NEEB, C., THUL, M., WEHN, N. Network-on-chip-centric approach to interleaving in high throughput channel decoders. *IEEE International Symposium Circuits and Systems*, 2005, pp. 1766 – 1769, ISBN 0-7803-8834-8.

[24] NĚMEC, K. *Kódové zabezpečovací systémy*. Teze habilitační práce, ÚTKO FEI VUT Brno, Brno 1999, 34 s., ISBN 80-214-1312-3.

[25] NOVAIS, E., BARANGER, U. Decoherence by correlated noise and quantum error correction. *Physical Reviev Letters*, 2006, Vol. 97, Issue 4, pp. 1 - 4, ISSN 0031-9007.

[26] PINKR, J., POUPA, M. *Číslicové systémy a jazyk VHDL*. 1. vyd., BEN, Praha 2006, 352 s., ISBN 80-7300-198-5.

[27] RFC 3452 - *Forward Error Correction (FEC) Building Block*. Dokument dostupný na URL http://www.faqs.org/rfcs/rfc3452.html (květen2008).

[28] RON, M. *Introduction to coding theory*. Cambridge University Press, Cambridge 2006, 566 p., ISBN 0-521-84504-1.

[29] SHIINIZU, K., et al. Reconfigurable adaptive FEC system with interleaving. *In Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific*, 2005, Vol. 2, pp. 1252 - 1255, ISBN 0-7803-8736-8.

[30] SHU, L., COSTELLO, D. *Error Control Coding – Fundamentals and Applications.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1983, 603 p., ISBN 0-13-283796-X.

[31] STOERTE, C., MACHMERTH, M. Implementation of error decoders for A-VSB systems with additional use of transport stream information as forward error correction. *In Devices, Circuits and Systems, 2008. ICCDCS 2008. 7th International Caribbean Conference*, 2008, Cancun, pp. 1 - 4, ISBN 978-1-4244-1956-2.

[32] SÝKORA, J. *Modulace, kódování a zpracování signálu v MIMO rádiových komunikačních systémech*. ČVUT v Praze, Praha 2006, 34s., ISBN 80-01-03557-3.

[33] THUL, M., GILBERT, F., VOGT, T. A Scalable System Architecture for High-Throughput Turbo-Decoders. *Journal of VLSI Signal Processing Systems*, 2005, Vol. 39, Issue 1 - 2, pp. 63 – 77, ISSN 0922-5773.

[34] THUL, M., WEHN, N. FPGA implementation of parallel turbo-decoders. *17th Symposium on Circuits and Systems Design*, 2004, pp. 198 – 203, ISBN 1-58113-947-0.

[35] VITERBI, A. An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes. *IEEE J. Select. Areas Commun.*, 1998, Vol. 16, No. 2, pp. 260 – 264, ISSN 0733-8716.

[36] VLČEK, K. *Komprese a kódová zabezpečení v multimediálních komunikacích.* 2. vyd., BEN, Praha 2004, 258 s., ISBN 80-7300-134-9.

[37] VOZŇÁK, M., ZUKAL, D. *Hodnocení kvality hlasu v IP sítích*. Technická zpráva CESNET, 2005, č. 11, s. 1 – 2.

[38] VRBA, R., a kol. *Digitální obvody a mikroprocesory.* Skriptum ÚTKO FEKT VUT Brno, Brno 2003, 251 s., ISBN 80-214-2593-8.

[39] WILSON, G. *Digital Modulation and Coding, Englewood Cliffs*, Prentice - Hall, 1996, 712 p., ISBN 0-13-210071-1.

[40] Xilinx *Spartan-3 FPGA Family Data Sheet*. 2008, 216 p. Dokument dostupný na URL http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf (červenec 2008).

[41] YITANG, D., XIANGFEI, CH., JIE, S. High-performance, high-chip-count optical code division multiple access encoders-decoders based on a reconstruction equivalent-chirp technique. *Optics Letters*, 2006, Vol. 31, Issue 11, pp. 1618 - 1620, ISSN 0146-9592.

[42] ZHAN, X., FITZ, M. Space-Time Code Design with Continuous Phase Modulation. *IEEE J. Select. Areas Commun.*, 2003, Vol. 21, pp. 783 – 792, ISSN 0733-8716.

Vlastní publikační činnost související s disertační prací:

[43] ČÍKA, P., KŘIVÁNEK, V., KOTON, J. Samoopravné Reed-Solomonovy kódy. *Access Server*, 2006, roč. 2006, č. 10, s. 1 - 6, ISSN 1214-9675.

[44] KŘIVÁNEK, V., ČÍKA, P. Simulace shlukových chyb v Matlabu. *Elektrorevue - Internetový časopis* (http://www.elektrorevue.cz), 2006, roč. 2006, č. 35, s. 1 - 10, ISSN 1213-1539.

[45] KŘIVÁNEK, V., ČÍKA, P. Výběr nejvhodnějšího konvolučního kódu - I. *Access Server*, 2007, roč. 5, č. 3, s. 1 - 8, ISSN 1214-9675.

[46] KŘIVÁNEK, V., KYSELÁK, M. Výběr nejvhodnějšího konvolučního kódu - II. *Access Server*, 2007, roč. 5, č. 3, s. 1 - 6, ISSN 1214-9675.

[47] KŘIVÁNEK, V. Oprava shlukových chyb - Iwadariho kód. In *Elektrotechnika a informatika 2005*. Plzeň: Západočeská univerzita v Plzni, 2005. s. 45 - 48, ISBN 80-7043-374-4.

[48] KŘIVÁNEK, V. The Use of Matlab for the Simulation of the Burst Error Correction. *International Journal of Computer Science and Network Security*, 2006, Vol. 6, No. 7B, pp. 141 - 145, ISSN 1738-7906.

[49] KŘIVÁNEK, V. Návrh kodérů a dekodérů umožňující opravu shlukových chyb a jejich simulace. *Elektrorevue - Internetový časopis* (http://www.elektrorevue.cz), 2006, roč. 2006, č. 8, s. 1 - 9, ISSN 1213-1539.

[50] KŘIVÁNEK, V. Verification of the error-control security process by means of simulation. *International Transactions on Communication and Signal Processing*, 2006, Vol. 9, No. 1, pp. 128 - 136, ISSN 1738-9682.

[51] KŘIVÁNEK, V. Využití počítačové simulace pro výuku principů zabezpečovacích kódů. In *5. ročník konference Alternativní metody výuky*. GAUDEAMUS, Universita Hradec Králové: Univerzita Karlova v Praze, 2007, s. 1 - 4, ISBN 978-80-7041-129-2.

[52] KŘIVÁNEK, V. Computer Simulation Forward-Eror-Corection Principles. In *8th International Conference, Research in Telecommunication Technology RTT - 2007 Liptovský Ján, SR*. 2007, pp. 209 - 214, ISBN 978-80-8070-735-4.

[53] KŘIVÁNEK, V. Ochrana dat před shluky chyb, Berlekamp-Preparatův kód. *Elektrorevue - Internetový časopis* (http://www.elektrorevue.cz), 2007, roč. 9, č. 49, s. 1 - 7, ISSN 1213-1539.

[54] KŘIVÁNEK, V. Correction Error Data Rising in the Transmission Channel. *International Transaction on Computer Science and Engineering*, 2007, Vol. 43, No. 1, pp. 9 - 16, ISSN 1738-6438.

[55] KŘIVÁNEK, V. Simulace korekčních kódů - QoS. In *6. ročník konference Alternativní metody výuky*. Přírodovědecká fakulta UK, Praha: Univerzita Karlova v Praze, 2008, s. 1 - 6, ISBN 978-80-7041-454-5.

[56] KŘIVÁNEK, V. Enhancement in the Protection of Transmitted Data. *International Journal of Computer Science and Network Security*, 2008, Vol. 8, No. 7, pp. 95 – 98, ISSN 1738-7906.

[57] KŘIVÁNEK, V. Comparison of the convolutional codes design complexity. *International Transaction on Computer Science and Engineering*, 2008, Vol. 47, No. 1, pp. 43 - 48, ISSN 1738-6438.

Bibliografická citace práce:

KŘIVÁNEK, V. *Systémy realizace protichybového kódování*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008, 93 s. Vedoucí dizertační práce doc. Ing. Karel Němec, CSc.

CURRICULUM VITAE

Osobní údaje

Jméno:	Vítězslav Křivánek
Datum a místo narození:	1. června 1981 v Třebíči
E-mail:	krivan@feec.vutbr.cz, V.Krivanek@seznam.cz

Vzdělání

od 2005	**VUT v Brně** Fakulta elektrotechniky a komunikačních technologií Doktorské studium Obor: Teleinformatika Téma disertační práce: Systémy realizace protichybového kódování.
2000 - 2005	**VUT v Brně** Fakulta elektrotechniky a komunikačních technologií Magisterské studium Obor: Elektronika a sdělovací technika
1993 - 2000	Gymnázium Vídeňská, Brno (maturita)

Další vzdělání

2007	Osvědčení Hlavní vedoucí dětských táborů
2006	Vedoucí oddělení vědy a techniky pro každého na Junior DDM Brno – pedagogický pracovník
2005	Osvědčení Kurz vedoucích herních klubů
2004	Invention Machine - TRIZ Certificate
2003	Autodesk Academia Certificate - AutoCAD 2002
2002	Osvědčení Zdravotník zotavovacích akcí
2000	Osvědčení Vedoucí dětského kolektivu

Ostatní

- Účast na projektech:

  GAČR reg. č. GA102/03/0762 „Analýza přenosových parametrů xDSL systémů pomocí reálných přístupových sítí“ (spoluřešitel).

  NBÚ reg. č. ST200520005002 „Synchronizace blokových šifer pro modulární kryptografický systém pro verzi BRI ISDN a PRI ISDN“ (spoluřešitel).

  FRVŠ reg. č. 256/2006/F1a „Inovace způsobu uskutečňování přednášek předmětu Datová komunikace“ (spoluřešitel).

  MPO ČR reg. č. FT-TA3/001 „Výzkum a vývoj obousměrné komunikační technologie pro varování obyvatelstva“ (spoluřešitel).

FRVŠ reg. č. 993/2007/G1 „Simulace zabezpečovacích korekčních kódů a její začlenění do výuky předmětu Datová komunikace“ (spoluřešitel).

FRVŠ reg. č. 1663/2007/F1a „Podpora výuky předmětu zaměřeného na programování v jazyce C++ pomocí interaktivních kontrolních testů“ (spoluřešitel).

FRVŠ reg. č. 1388/2008/F1a „Zdokonalení výuky zabezpečování přenosu informace“ (spoluřešitel).

FRVŠ reg. č. 1775/2008/G1 „Zvýraznění problematiky kvality služeb v předmětu Služby telekomunikačních sítí“ (spoluřešitel).

NPV II. reg. č. 2C08002 „KAAPS „Výzkum univerzální a komplexní autentizace a autorizace pro pevné a mobilní počítačové sítě“ (spoluřešitel).

- Autor, resp. spoluautor 22 publikací vztahujících se k řešené problematice.