VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY

DEPARTMENT OF MICROELECTRONICS

KODÉR A DEKODÉR SAMOOPRAVNÉHO KÓDU PRO PROGRAMOVATELNÉ PAMĚTI TYPU ROM

AN ENCODER AND DECODER OF AN ERROR-CORRECTION CODE FOR PROGRAMMABLE READ-ONLY MEMORIES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE Jan Bareš

AUTHOR

VEDOUCÍ PRÁCE Ing. Martin Štáva, Ph.D.

SUPERVISOR

BRNO 2016

Bakalářská práce

bakalářský studijní obor **Mikroelektronika a technologie**

Ústav mikroelektroniky

Student: Jan Bareš ***ID:*** 161573

Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Kodér a dekodér samoopravného kódu pro programovatelné paměti typu ROM

POKYNY PRO VYPRACOVÁNÍ:

Analyzujte dostupné samoopravné kódy (ECC) a jejich parametry. Vyberte samoopravný kód schopný detekovat dvojnásobnou chybu a opravit jednonásobnou chybu v každém slově paměti. Pro vybraný samoopravný kód vytvořte aplikaci generující kodér (pro kódování slov při zápisu do paměti) a dekodér (pro dekódování slov při čtení z paměti), které budou popsány syntetizovatelným kódem v jazyce VHDL na úrovni meziregistrových přenosů (RTL). Šířka slova paměti je volitelná. Dále navrhněte a v jazyce VHDL popište model programovatelné paměti typu ROM s možností vkládání chyb. V digitálním simulátoru proveďte automatickou validaci vygenerovaného kodéru a dekodéru.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 8.2.2016 ***Termín odevzdání:*** 2.6.2016

Vedoucí práce: Ing. Martin Šťáva, Ph.D.

Konzultant bakalářské práce: Ing. Miroslav Kašša, ON Design Czech, s.r.o. Brno

doc. Ing. Jiří Háze, Ph.D., *předseda oborové rady*

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Práce se zabývá teorií kódování, analyzuje současné skupiny samoopravných kódů a popisuje vlastnosti a parametry vybraných zástupců těchto skupin. Na základě daných kritérií vybírá porovnáním těchto parametrů a vlastností rozšířený Hammingův kód jako vhodný kód pro zabezpečení paměti typu read-only-memory (ROM). Práce popisuje návrh syntetizovatelných modulů kodéru a dekodéru v jazyku VHDL. Dále vysvětluje princip činnosti vytvořené aplikace, která je schopna generovat tyto syntetizovatelné moduly. Pro ověření generovaných modulů vytváří verifikační prostředí, jehož součástí je i model paměti typu ROM, umožňující zápis libovolné chybové hodnoty do paměti. Na závěr provádí automatickou verifikaci generovaných modulů kodéru a dekodéru pro různé šířky vstupního informačního vektoru.

KLÍČOVÁ SLOVA

Samoopravný kód, paměť ROM, SEC-DED, BCH kód, rozšířený Hammingův kód, optimalizace kontrolní matice, generování kodéru a dekodéru, verifikační prostředí, verifikace

.

ABSTRACT

This work deals with theory of coding, analyses current groups of error correction codes and describes features and parametres of chosen representatives of these groups. By comparing these parametres along with given criteria it choses extended Hamming code as suitable code for securing read-only-memories (ROM). For this code it choses way of realization of synthetisable modules of coder and decoder and describes their design. The work describes design of synthetizable modules of coder and decoder in VHDL. Then it explains functionality of created application which is able to generate these synthetisable modules. For verification of generated modules it creates authentication environment. Part of this environment is also model of ROM allowing writing of any error value into the memory. In the end it automatically verifies generated modules of coder and decoder with various width of input information vector.

KEYWORDS

Error Correction Code, ROM, SEC-DED, BCH code, extended Hamming dode, parity-check matrix optimalization, verification environment.

BIBLIOGRAFICKÁ CITACE DÍLA:

BAREŠ, J. *Kodér a dekodér samoopravného kódu pro programovatelné paměti typu ROM*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2016. 61 s. Vedoucí bakalářské práce Ing. Martin Šťáva, Ph.D..

PROHLÁŠENÍ AUTORA O PŮVODNOSTI DÍLA:

Prohlašuji, že svou bakalářskou práci na téma Kodér a dekodér samoopravného kódu pro programovatelné paměti typu ROM jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujícího zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 2. června 2016 ....

podpis autora

PODĚKOVÁNÍ:

Děkuji vedoucímu své bakalářské práce Ing. Martinu Šťávovi, Ph.D., za účinnou metodickou, pedagogickou a odbornou pomoc, za věnovaný čas, vstřícnost a ochotu pomoci i za veškeré cenné rady při zpracování mé bakalářské práce. Dále děkuji Ing. Miroslavu Kassovi ze spolupracující firmy ON Design Czech, s.r.o., za poskytnutí příležitosti ke zpracování mé práce, za trpělivost při vysvětlování, za všechny cenné připomínky k práci a za veškerou odbornou pomoc. Děkuji také mé dívce, mojí rodině a přátelům, kteří mě v průběhu psaní práce podporovali. V neposlední řadě děkuji Bohu za jeho dobrotivost, pomoc i oporu při zpracovávání této práce.

OBSAH

ÚVOD 8

TEORETICKÁ ČÁST 9

1 ÚVOD DO KÓDOVÁNÍ 9

1.1 Kódování a dekódovaní 9

1.2 Kód 9

2 ZÁKLADY LINEÁRNÍ ALGEBRY 10

2.1 Grupa 10

2.2 Těleso 11

2.3 Okruh 11

2.4 Vlastnosti těles 12

2.5 Vektorový prostor 13

3 LINEÁRNÍ KÓDY 14

3.1 Důležité pojmy 14

3.2 Parametry kódů 17

3.3 Vlastnosti kódu 18

3.4 Zástupci lineárních kódů 20

4 CYKLICKÉ KÓDY 23

4.1 Důležité pojmy 23

4.2 Vlastnosti cyklických kódů 27

4.3 Zástupci cyklických kódů 28

5 SROVNÁNÍ 31

PRAKTICKÁ ČÁST 34

6 ROZBOR SOUČASNÉHO STAVU 34

6.1 Kodéry a dekodéry Hammingova kódu 34

6.2 Configurable Hamming Generator 34

6.3 Projekt „UMEL_ŠB“ 35

7 KODÉR A DEKODÉR HAMMINGOVA KÓDU 35

7.1 Vlastní řešení problému 35

7.2 Syntetizovatelné moduly v jazyku VHDL 36
7.2.1 Kodér Hammingova kódu 36
7.2.2 Dekodér Hammingova kódu 38
7.2.3 Knihovní balíky *hamm_rom_xb_pkg* 40

7.3 Aplikace generující syntetizovatelné moduly 40

8 VERIFIKAČNÍ PROSTŘEDÍ 42

8.1 Struktura prostředí 42

8.2 Zásady psaní zdrojových kódů 45

8.3 Model programovatelné paměti typu ROM 46

8.4 Model řídící jednotky paměti *rom_master* 48

8.5 Pomocné moduly a knihovní balíky 50

9 VERIFIKACE 51

9.1 Zásady pro verifikaci 51

9.2 Verifikační plán 52

9.3 Modul testcase 54

9.4 Výsledky verifikace 56

ZÁVĚR 57

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK 59

LITERATURA 60

ÚVOD

V minulých letech docházelo k neustálým pokrokům v polovodičové technologii, které vedly ke zmenšení součástek na čipu a zároveň k jejich větší integraci. V důsledku toho pokroku však v těchto obvodech, mezi které patří i paměťové moduly, také dochází k většímu počtu chyb při výrobě a obvody jsou stále více náchylné na elektrické, teplotní i mechanické namáhání. Mimo to mohou v obvodech vznikat také přechodné chyby, např. následkem dopadu alfa částic nebo nedodržením provozních teplot. Jeden z efektivních způsobů, jak odstranit vliv těchto chyb, je použití samoopravných kódů.

Samoopravný kód zavádí tzv. informační redundanci, tj. k přenášené informaci jsou přidány znaky, které nesou doplňkovou informaci, díky níž je možné detekovat a případně opravit chyby. Vždy je snaha najít kód opravující žádaný počet chyb, který při určité délce přenášené informace má co nejmenší redundanci.

Cílem této práce je analyzovat dostupné samoopravné kódy, které jsou používány pro opravu chyb v pamětech typu ROM a vybrat samoopravný kód, který je vhodný pro tyto typy pamětí a zároveň dokáže opravit jednonásobnou a detekovat dvojnásobnou chybu. Tato práce si nečiní nárok na popsání všech používaných samoopravných kódů pro zabezpečení pamětí. Je zaměřena na kódy, které jsou nejběžnější při zabezpečování informace v pamětech a hlavně nejvýhodnější. Tento výběr je poté upraven tak, aby vybrané kódy splňovaly zadané kritérium.

V teoretické části práce je vysvětlena nezbytná terminologie a teorie kódovaní, založená na algebraických strukturách. V hlavní části se text zaměřuje na dvě skupiny kódů – lineární kódy a cyklické kódy. Pro každou z těchto skupin je vysvětleno nezbytné názvosloví, popsány vlastnosti těchto kódů a uvedeni významní zástupci těchto skupin. V závěru teoretické části jsou shrnuty a porovnány vlastnosti uvedených zástupců kódů a na základě daných kritérií vybrán rozšířený Hammingův kód jako nejvhodnější kód.

V praktické části práce je rozebrán návrh syntetizovatelných modulů kodéru a dekodéru, jež byly popsány parametricky v jazyku VHDL i popis vytvořené aplikace v jazyku C, která dokáže tyto syntetizovatelné moduly generovat. Dále se text zaměřuje na popis verifikačního prostředí, jež bylo vytvořeno k automatické verifikaci generovaných modulů kodéru a dekodéru. Jsou zde popsány modely verifikačního prostředí, včetně modelu paměti typu ROM, umožňující zápis libovolné chybové hodnoty do paměti. V závěru práce jsou podány výsledky automatické verifikace vybraných syntetizovatelných modulů kodérů a dekodérů pro různé šířky vstupních informačních vektorů.

TEORETICKÁ ČÁST

1 ÚVOD DO KÓDOVÁNÍ

V číslicových systémech je třeba počítat s tím, že při přenosu informace může dojít k chybě, která vede k přijmutí zprávy rozdílné od původní zprávy. Jednou z nejdůležitějších metod zabezpečení číslicových systémů proti vlivu chyb je použití bezpečnostních kódů.

Bezpečnostní kód je takový kód, ve kterém je zavedena tzv. *informační redundance* – ke znakům přenášené informace jsou přidány nadbytečné znaky, které nenesou žádnou informaci, ale slouží ke kontrole správnosti [2]. Tyto znaky mohou být zvoleny tak, že se na výstupu mohou objevovat pouze některé kombinace (tzv. kódová slova) logických hodnot z množiny všech možných kombinací. Pokud dojde při přenosu k chybě, změní se hodnota některých přenášených znaků a může tak dojít k přijetí slova, které není kódové. V takovém případě jsme schopni chybu detekovat a za určitých podmínek i opravit [1].

Pro správný vhled do tématiky je potřebná znalost terminologie, která je předkládána na několika následujících stranách. Definice jednotlivých pojmů jsou prokládány příklady pro snazší pochopení problematiky.

1.1 Kódování a dekódovaní

Kódování a dekódování. *Kódováním* rozumíme postup, který každému prvku konečné množiny A přiřazuje slovo v konečné množině B. Pokud se jedná o opačný postup, nazýváme jej *dekódování*. Kódování nemusí být jednoznačné – jednomu prvku množiny A může odpovídat více prvků množiny B. Naproti tomu u dekódování je jednoznačnost vyžadována. [1]

Slovo. *Slovo* je konečná posloupnost prvků z množiny T – tedy posloupnost $a_1 a_2 \dots a_n$ kde $a_i \in T$. Takové slovo má *délku n* a množinu T označujeme jako *abecedu*. Slovo může být chápáno také jako vektor n-rozměrného vektorového prostoru [2].

Množina T^n: Množina všech slov délky n, kde $n \in \mathbb{P}$ je označována jako T^n, tedy [2]:

$$T^n = \{t_1, t_2 \dots t_n \mid t_i \in T \; pro \; i = 1,2,\dots,n\}$$

1.2 Kód

Kód. *Kód* je libovolná neprázdná množina slov (kódových vektorů) [2].

Blokový kód. Kód, jehož všechna slova mají stejnou délku n ($n \in \mathbb{N}$), bývá označován jako *blokový kód* [1]. V bezpečnostních kódech bývají tyto kódy používané nejčastěji, proto označením kód bude vždy rozuměn právě blokový kód [3].

Kódová a nekódová slova. Pokud délka slov kódu K je rovna n, potom K je podmnožinou množiny Tn všech slov délky n a tyto slova dělíme na *kódová slova* (v K) *a nekódová slova* (v $T^n - K$) [2].

Binární kódování. Nejdůležitější případ kódování v číslicových systémech je binární kódování, při kterém jsou kódová slova vyjádřena jako posloupnost prvků 0 a 1. Tyto znaky reprezentují fyzikální veličiny, které nabývají dvou diskrétních hodnot nebo hodnot ze dvou vzájemně disjunktních intervalů (takových intervalů, které nemají společný prvek). Pokud tedy nebude řečeno jinak, bude v následujícím textu pod pojmem kód vždy uvažován právě binární kód [1].

Kontrolní a informační znaky. Znaky každého kódu můžou být rozděleny na *informační*, které lze libovolně zvolit a *kontrolní*, které jsou zcela určeny volbou informačních znaků. Tedy pro blokový kód K délky n, který je podmnožinou množiny T^n všech slov délky n platí: pokud můžeme z n znaků kódového slova libovolně volit k informačních znaků, máme předpis, který libovolné zdrojové slovo délky k převede na kódové slovo délky n. Právě takovému předpisu se říká kódování informačních znaků a výsledný kód K má tedy délku n, z čehož je k znaků informačních a $(n - k) = r$ znaků kontrolních [2].

Příklad. Jako příklad můžeme ukázat binární kód celkové parity délky n. Je to jeden z nejobvyklejších bezpečnostních kódů vůbec. K binárnímu slovu přidáme jeden kontrolní znak tak, aby výsledné slovo mělo sudou paritu (sudý počet jedniček). Tím vznikne binární kód délky n všech slov sudé parity. Chceme-li např. přenášet $k = 3$ informační bity, vznikne kód délky $n = 3 + 1 = 4$ $(k + r = n)$. Prvky tohoto kódu tvoří 8 kódových slov o 4 znacích.

$K = \{0000,1100,1010,0110,1001,0101,0011,1111\}$

Tento kód je podmnožinou T^4 obsahující všechny slova délky 4. Např. slovo 0000 je kódové slovo, zatímco slovo 0001 je nekódové slovo [2].

2 ZÁKLADY LINEÁRNÍ ALGEBRY

V teorii kódování budeme potřebovat zavést algebraické operace sčítání a násobení na abecedě T (V našem případě uvažujeme vždy binární kódy, tedy $T = \{0, 1\}$). Potom T se stane tělesem a T^n se stane n-rozměrným lineárním prostorem. Pro srozumitelnost výkladu dalšího textu jsou zde shrnuta některá fakta lineární algebry a připomenuty základní algebraické struktury, se kterými budeme pracovat: grupy, tělesa, okruhy a vektorové prostory [2].

2.1 Grupa

Grupa. *Grupa* je množina prvků $G = (\mathbb{G}, *)$ s **jednou** operací *, splňující:

G1) $a * b = ab$ je prvkem množiny G (uzavřenost grupy)

G2) Pro $a, b, c \in G$ platí $a * (b * c) = (a * b) * c$ (asociativní zákon)

G3) G obsahuje neutrální prvek 1 takový, že $(1a = a)$ a $(a1 = a)$

G4) Pro každý prvek $a \in G$ existuje inverzní prvek a^{-1} takový, že ($aa^{-1} = 1$) a ($a^{-1}a = 1$)

Pokud je grupa komutativní, tj $a * b = b * a$, označujeme grupovou operaci + a nazýváme ji *sčítání* a inverzní prvek označujeme jako – a a nazýváme ho *opačný prvek* [2].

2.2 Těleso

Těleso. *Těleso* je množina prvků $T = (T, +, *)$ se **dvěma** operacemi + a *, splňující:

T1) (T, +) je komutativní grupa s neutrálním prvkem 0.

T2) (T – {0}, *) je komutativní grupa s neutrálním prvkem 1. (Pro obě operace tedy platí uzavřenost, asociativní i distributivní zákon)

T3) Obě operace jsou svázány distributivním zákonem, tj. platí $a* (b + c) = (a * b) + (a * c)$

T4) T obsahuje neutrální prvky 0,1 množiny T takové, že $(a + 0) = a$ a $(a * 1) = a$ (pro všechna $a \in T$)

T5) Každý prvek a množiny T má opačný prvek – a takový, že $a – a = 0$.

T6) Každý prvek $a \neq 0$ množiny T má inverzní prvek, tj. prvek a^{-1} takový, že $aa^{-1} = 1$.

V tělese tedy máme operace sčítání, odčítání, násobení a dělení (nenulovými prvky) splňující asociativní, komutativní a distributivní zákony [2].

Galoisova těleso. Tělesa s konečným počtem prvkům (tzv. konečná tělesa) se nazývají *Galoisova tělesa* a značí se $GF(q)$, kde q označuje počet prvků. Lze dokázat, že existují pouze konečná tělesa $GF(q)$, kde q je mocnina prvočísla, tedy $q = p^t$, kde p je prvočíslo a t přirozené číslo. Naopak, jestliže p je prvočíslo a t přirozené číslo, pak vždy existuje konečné těleso T, jehož počet prvků m je roven p^t. Konečná tělesa jsou (až na izomorfismus) jednoznačně určena počtem svých prvků [2].

2.3 Okruh

Okruh. *Okruh* je množina prvků $T = (T, +, *)$ se dvěma operacemi + a *, splňující axiómy tělesa T1) až T5). Prvky okruhu tedy nemusí mít inverzní prvek. Příkladem je okruh $\mathbb{Z}$ celých čísel (který není tělesem, protože žádný prvek kromě 1 a -1 nemá inverzní prvek) [2].

Třída modulo *q*. Zvolíme-li v okruhu T libovolný prvek a, potom *třídou modulo q* daného prvku a v T nazýváme množinu prvků, které se od a liší o násobek prvku q, tedy množinu prvků, které mají stejný zbytek po dělení číslem q. Třídy označujeme složenými závorkami, ve kterých je napsán libovolný reprezentant třídy [2].

Příklad 4.3.1. V okruhu $\mathbb{Z}$ všech celých čísel existují právě dvě třídy modulo 2 a to třída {0} všech sudých čísel, které se od sebe navzájem liší o násobek 2 a třída {1} všech lichých čísel, které se od sebe navzájem liší o násobek 2 [2].

2.4 Vlastnosti těles

Izomorfismus. Dvě tělesa *T* a *T'* jsou *izomorfní*, jestliže existuje vzájemně jednoznačné zobrazení

$$t: T \to T'$$

zachovávající algebraické operace. Tedy dvě tělesa, jenž jsou izomorfní, jsou v podstatě totožná: jejich prvky mají pouze jiné označení a pokud „přejmenujeme" prvek *a* tělesa *T* na prvek *a'* tělesa *T'*, potom toto přejmenování nemění algebraickou strukturu [2].

Zbytkové třídy modulo *q*. Těleso $GF(q)$ je izomorfní s tělesem celých čísel $\mathbb{Z}$ modulo q. V tělese $\mathbb{Z}$ modulo q je q prvků: 0, 1, …, q-1. Operace sčítání a násobení se provádějí stejně jako v aritmetice, přesáhne-li však výsledek v absolutní hodnotě číslo (q-1), bere se za výsledek operace sčítání a násobení zbytek po dělení číslem q. Číslo q se nazývá modul. Proto se prvkům tělesa $\mathbb{Z}$ modulo q říká *zbytkové třídy modulo q*.

V číslicových systémech hraje důležitou roli těleso $GF(2)$ s prvky {0, 1}, kde sčítání a násobení se provádí modulo 2. Sčítání v $GF(2)$ je možné jednoduše technicky realizovat logickým obvodem nonekvivalence, násobení lze realizovat logickým součinem. Sčítání a odčítání v $GF(2)$ splývá [1].

+	0	1
0	0	1
1	1	0

*	0	1
0	0	0
1	0	1

Obr. 1: Pravidla sčítání a násobení v tělese $GF(2)$ [1]

Zbytkové třídy polynomů modulo $q(x)$. Konečná tělesa $GF(p^n)$, kde $p^n = q$, jsou izomorfní s tělesem *zbytkových tříd polynomů modulo* $q(x)$, kde $q(x)$ je *nerozložitelný* (*ireducibilní*) *polynom* n-tého stupně s koeficienty z $GF(p)$. (Nerozložitelný polynom n-tého stupně je takový, který není dělitelný polynomem z $GF(q)$ stupně nižšího než n a vyššího než 0). Polynomu $q(x)$ se říká modulový, neboť operace v $GF(p^n)$ dávají za výsledek zbytek po dělení tímto polynomem [1].

Příklad 4.4.1. Prvky tělesa $GF(2^2)$ můžeme reprezentovat zbytkovými třídami polynomů s koeficienty z $GF(2)$, kde polynomy se berou modulo x^2+x+1. Prvky tohoto tělesa jsou tvořeny zbytky po dělení {0}, {1}, $\{x\}$, $\{x+1\}$. (Složenými závorkami zde označujeme zbytkové třídy polynomů) [1].

Modulový polynom je sice v $GF(p)$ nerozložitelný, ale v $GF(p^n)$ má n kořenů. Jedním z jeho kořenů je vždy $\alpha = \{x\}$. Lze dokázat, že zbývající kořeny $g(x)$ mají tvar [1]:

$$\alpha^p, (\alpha^p)^2, \dots, (\alpha^p)^{n-1}$$

2.5 Vektorový prostor

Vektorový prostor. *Vektorový* (též *lineární*) *prostor* nad tělesem T je množina $V = (\mathbb{V}, +)$ s vnitřní operací + (sčítání) a „vnější“ operací · (skalární násobení) takovou, že pro každé těleso $x \in V$ a každý skalár $r \in T$ je definován prvek $r \cdot x = rx$ z množiny V a přitom pro každé $x, y \in V$ a každá čísla $r, s, \in T$ je splněno: [2]:

V1) $(V, +)$ je komutativní grupa

V2) $rx \in V$

V3) $r \cdot (x + y) = r \cdot x + r \cdot y$

V4) $(r + s) \cdot x = r \cdot x + s \cdot x$,

V5) $(r \cdot s) \cdot x = r \cdot (s \cdot x)$

V6) $1 \cdot x = x$

V teorii kódování se nejčastěji pracuje s vektorovým prostorem ${}^nV_q$, kde vektory zde budou zapisovány jako n-tice prvků $GF(q)$ tj. $\boldsymbol{u} = (u_0, u_1, \dots, u_{n-1})$. V případě binárního kódování je $q = 2$ a proto pracujeme s vektorovým prostorem ${}^nV_2$, tedy s množinou všech slov délky n, které jsou tvořeny prvky z $GF(2) = \{0, 1\}$ [1].

Sčítání vektorů $\boldsymbol{u} = (u_0, u_1, \dots, u_{n-1})$ a $\boldsymbol{v} = (v_0, v_1, \dots, v_{n-1})$ je definováno jako:

$$\boldsymbol{u} + \boldsymbol{v} = (u_0 + v_0, u_1 + v_1, \dots, u_{n-1,} + v_{n-1})$$

násobení prvkem $a \in GF(q)$

$$a \cdot \boldsymbol{u} = (a \cdot u_0, a \cdot u_1, \dots, a \cdot u_{n-1})$$

a skalární součin vektorů, jehož výsledek patří do $GF(q)$

$$\boldsymbol{u} \cdot \boldsymbol{v} = (u_0 \cdot v_0 + u_1 \cdot v_1 + \dots + u_{n-1} \cdot v_{n-1})$$

Je-li skalární součin roven 0, říkáme, že $\boldsymbol{u}$ a $\boldsymbol{v}$ jsou *ortogonální* [1].

Lineární nezávislost. Buď V vektorový prostor. Vektory $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n \in V$ se nazývají *lineárně nezávislé*, jestliže pro každé $a_1, a_2, \dots, a_n \in GF(q)$ platí [8]:

$$a_1\boldsymbol{v_1} + a_2\boldsymbol{v_2} + \cdots + a_n\boldsymbol{v_n} = \boldsymbol{o} \quad \Rightarrow \quad a_1 = a_2 = \cdots = a_n = 0.$$

V opačném případě řekneme, že vektory $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n$ jsou *lineárně závislé*. Zjednodušeně lze říci, že vektory jsou lineárně nezávislé, pokud nelze některý z nich vyjádřit jako lineární kombinaci ostatních (součet násobků ostatních vektorů) [2].

Báze. Libovolná lineárně nezávislá uspořádaná n-tice $\beta = (\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_n)$ vektorů z V, která generuje celý prostor V, se nazývá *báze* vektorového prostoru V [2].

Dimenze. Číslo n se nazývá *dimenze* prostoru ${}^nV_q$. Lze v něm tedy najít n lineárně nezávislých vektorů - vektorů báze. Každý vektor $\boldsymbol{v}$ z ${}^nV_q$ lze vyjádřit jako lineární kombinaci vektorů báze [1].

Konečné těleso $GF(2^n)$ (reprezentující množinu všech binárních slov délky n) je vektorovým prostorem dimenze n nad $GF(2)$. Všechny prvky $GF(2^n)$ lze tedy vyjádřit lineární kombinací n lineárně nezávislých prvků. [1]

Příklad: Konečné těleso $GF(2^3)$ je vektorovým prostorem ${}^3V_2$, který má dimenzi n = 3 a tvoří množinu všech binárních slov délky n = 3. Takových slov je právě $2^3 = 8$, a sice $GF(2^3)$ = {000,001,010,011,100,101,110,111} Báze tohoto prostoru $\beta = (\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3)$ je:

$\boldsymbol{b}_1 = (100)$

$\boldsymbol{b}_2 = (010)$

$\boldsymbol{b}_3 = (001)$

Bázové vektory jsou lineárně nezávislé, neboť žádný z nich není možné získat lineární kombinací ostatních. Lineární kombinací těchto bázových vektorů ale můžeme vyjádřit jakýkoliv vektor prostoru ${}^3V_2$. Například vektor $\boldsymbol{v} = (101)$ je roven součtu $\boldsymbol{b}_1$ a $\boldsymbol{b}_3$.

3 LINEÁRNÍ KÓDY

3.1 Důležité pojmy

Lineární kód. Binární kód K se nazývá *lineární*, jestliže je podprostorem lineárního prostoru ${}^nV_2$, tj. jestliže součet dvou kódových slov je kódové slovo [2].

Grupový kód (n, k). Ve vektorovém prostoru ${}^nV_2$ všech binárních slov délky n, který je n-dimenzionálním prostorem nad $GF(2)$, lze volit řadu k-dimenzionálních podprostorů ${}^kV_2$, $k \leq n$. Každý takový podprostor nazýváme *grupovým kódem* (n, k), kde písmenem n je označována délka kódu a písmenem k je označován počet informačních bitů. Pokud nebude stanoveno jinak, budou písmeny n a k v celém zbytku textu označovány právě tyto pojmy [1].

Kódová a nekódová slova. Volbou prostoru ${}^kV_2$ se množina všech n-bitových vektorů rozpadá na množinu n-bitových vektorů z ${}^kV_2$, tzv. *kódových vektorů* (*slov*), a ostatních bitových vektorů tzv. *nekódových slov*, které do ${}^kV_2$ nepatří [1].

Generující matice *G*. Grupový kód je vektorovým prostorem, proto jej můžeme zadat pomocí k n-bitových vektorů báze. Zápisem vektorů pod sebe vznikne tzv. *generující matice* $\boldsymbol{G}$, která je typu $[k \times n]$ [1].

$$G = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \quad (3.1)$$

Tedy matice $\boldsymbol{G}$ je generující matice daného kódu, jestliže

a) Její řádky jsou tvořeny kódovými slovy

b) Každé kódové slovo můžeme získat jako lineární kombinaci řádků

c) Vektory, tvořící řádky matice jsou lineárně nezávislé, takže hodnost matice $\boldsymbol{G}$ je rovna počtu informačních bitů k.

Právě počet řádků udává dimenzi kódu, tedy počet jeho informačních bitů a tím při dané délce kódu n i jeho počet kontrolních bitů a minimální kódovou vzdálenost [2].

Kontrolní matice *H*. Druhý způsob zadání (volby) vektorového podprostoru ${}^{k}V_2$ je pomocí tzv. *kontrolní matice* $\boldsymbol{H}$ typu $[(n\text{-}k) \times n]$, která je tvořena soustavou homogenních lineárních rovnic. To znamená, že matice $\boldsymbol{H}$ z prvků abecedy T je kontrolní maticí kódu délky n, právě když platí: Slovo $\boldsymbol{v} = (v_0\, v_1\, \ldots\, v_{n\text{-}1})$ je kódovým slovem, právě když

$$\boldsymbol{H} \cdot \boldsymbol{v}^{\boldsymbol{T}} = \boldsymbol{H} \cdot \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \boldsymbol{0}^{\boldsymbol{T}} \quad (3.2)$$

kde $\boldsymbol{v}^{T}$ značí transpozici vektoru $\boldsymbol{v}$ a maticový součin $\boldsymbol{H} \cdot \boldsymbol{v}^{T}$ se řídí pravidly z $GF(2)$ tj. výsledek operace sčítání i násobení se bere modulo 2. Právě tímto způsobem je možné odhalit nekódové slovo a tím i chybu v přenosu. Protože generující matice je vždy tvořena kódovými vektory, musí také platit [2]:

$$\boldsymbol{H} \cdot \boldsymbol{G}^{T} = \boldsymbol{0}^{T} \quad (3.3)$$

Systematický kód. Systematický kód je takový lineární kód, který má kontrolní matici ve tvaru

$$\boldsymbol{G} = [\boldsymbol{E} \mid \boldsymbol{B}] \quad (3.4)$$

Kde písmenem $\boldsymbol{E}$ označujeme jednotkovou matici o typu $[k \times k]$. U takových kódů tedy tvoří prvních k souřadnic kódového vektoru nezměněné informační bity a zbylých $r = n - k$ bitů je kontrolních [2].

Ekvivalence. Dva blokové kódy K a K' délky n jsou *ekvivalentní*, jestliže kód K' vznikne pouhou permutací pořadí znaků v kódových slovech. (Kód K' tedy vznikne tak, že v každém z kódových slov kódu K přemístím vždy prvky a_i za prvky a_j.) [2].

Věta 3.1.1. Každý lineární kód je ekvivalentní se systematickým lineárním kódem. Důkaz tohoto tvrzení můžeme najít např. v [2. s 49].

Generující matici nesystematického binárního kódu je tak možné vždy převést pouze elementárními úpravami na matici typu (3.4). Pokud má generující matice tento tvar, pak kontrolní matice má tvar:

$$\boldsymbol{H} = [\boldsymbol{B}^T \mid \boldsymbol{E}'] \tag{3.5}$$

kde $\boldsymbol{B}^T$ je transponovaná matice $\boldsymbol{B}$ (tedy matice, jež vznikne vzájemnou výměnou řádků a sloupců matice $\boldsymbol{B}$) a $\boldsymbol{E}'$ je jednotková matice [2].

Příklad: *„Koktavý kód"* je typ kódu, ve kterém se každý znak opakuje dvakrát po sobě. Má tedy polovinu informačních a polovinu kontrolních znaků. Např. kód délky $n = 6$ s $k = 3$ informačními znaky má kódování definované předpisem:

$$\boldsymbol{\varphi}(\boldsymbol{v_1 v_2 v_3}) = \boldsymbol{v_1 v_1 v_2 v_2 v_3 v_3} \tag{3.6}$$

A generující matici proto můžeme zvolit ve tvaru:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Jakékoliv slovo, které je definované předpisem (3.6) můžeme získat jako lineární kombinaci řádků generující matice. Takovýto kód není systematický, neboť jeho kódová slova nemají na prvních $k = 3$ souřadnicích informační bity, které by byly následovány $r = 3$ kontrolními bity. Přehodíme-li druhý a čtvrtý sloupec, následně třetí a šestý sloupec generující matice, tj. provedeme-li permutaci $\pi = [1, 4, 6, 2, 5, 3]$ dostaneme systematický kód K', který je ekvivalentní s koktavým kódem. Tento ekvivalentní systematický kód má pak kódování definované předpisem:

$$\boldsymbol{\varphi}(\boldsymbol{v_1 v_2 v_3}) = \boldsymbol{v_1 v_2 v_3 v_1 v_3 v_2}$$

A jeho generující matice má tvar:

$$\boldsymbol{G}' = [\boldsymbol{E} \mid \boldsymbol{B}] = \begin{bmatrix} 1 & 0 & 0 & | & 1 & 0 & 0 \\ 0 & 1 & 0 & | & 0 & 0 & 1 \\ 0 & 0 & 1 & | & 0 & 1 & 0 \end{bmatrix}$$

Proto jeho kontrolní matice bude mít tvar:

$$\boldsymbol{H}' = [\boldsymbol{B}^T \mid \boldsymbol{E}'] = \begin{bmatrix} 1 & 0 & 0 & | & 1 & 0 & 0 \\ 0 & 0 & 1 & | & 0 & 1 & 0 \\ 0 & 1 & 0 & | & 0 & 0 & 1 \end{bmatrix}$$

Kód K' vznikl z koktavého kódu přehozením druhého a čtvrtého sloupce a třetího a šestého sloupce. Jestliže provedeme inverzní permutaci, tedy přehodíme-li sloupce zpět, dostaneme kontrolní matici koktavého kódu:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Pro ověření, zda je např. slovo $\boldsymbol{v} = (111111)$ kódové, můžeme využít rovnice (3.2). Tedy:

$$\boldsymbol{H} \cdot \boldsymbol{v}^T = \boldsymbol{H} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Tedy slovo $\boldsymbol{v} = (111111)$ je kódové.

3.2 Parametry kódů

U každého kódu budeme sledovat několik nejdůležitějších parametrů – délku kódu, počet informačních bitů a počet kontrolních bitů, informační poměr a minimální kódovou vzdálenost. První čtyři pojmy již byly definovány dříve, zde je pouze jejich připomenutí. Pro definování paramentu minimální kódové vzdálenosti jsou zde navíc zavedeny dva pojmy, které jsou taktéž důležité z hlediska detekce nezávislých chyb, a sice *Hammingova váha a Hammingova vzdálenost.* V této podkapitole budeme kódová slova délky n brát jako vektory n-rozměrného vektorového prostoru [1].

Délka kódu. Počet souřadnic kódového vektoru blokového kódu označujeme jako délku kódu n.

Počet informačních bitů. Počet bitů kódového vektoru, které obsahují zdrojovou informaci a jsou nezávislé na ostatních bitech, je označován jako počet informačních bitů k.

Počet kontrolních bitů. Počet bitů kódového vektoru, které neobsahují zdrojovou informaci a jsou zcela určeny volbou informačních znaků, je označován jako počet kontrolních bitů r.

Informační poměr. Poměr počtu informačních bitů ku celkovému počtu bitů je označován jako informační poměr R [2].

Hammingova váha. *Hammingova váha* vektoru $\boldsymbol{v}$ označuje počet jedničkových (obecně nenulových) souřadnic binárního vektoru a značí se $w(\boldsymbol{v})$ [1].

Hammingova vzdálenost: *Hammingova vzdálenost* $d(\boldsymbol{u}, \boldsymbol{v})$ udává počet souřadnic, ve kterých se liší dva binární vektory $\boldsymbol{u}$ a $\boldsymbol{v}$ o stejném počtu souřadnic. Pojmem *Kódová vzdálenost* pak označujeme Hammingovu vzdálenost mezi dvěma kódovými slovy. Pro dva binární vektory platí, že jejich Hammingova vzdálenost je rovna Hammingově váze součtu těchto vektorů *mod* 2 po souřadnicích tj. $d(\boldsymbol{u}\ \boldsymbol{v}) = w(\boldsymbol{u} + \boldsymbol{v})$ [1].

Příklad: Máme určit Hammingovu váhu a vzdálenost vektorů $\boldsymbol{u} = (0111)$ a $\boldsymbol{v} = (1100)$. Platí, že $w(\boldsymbol{u}) = 3$, $w(\boldsymbol{v}) = 2$, $\boldsymbol{u} + \boldsymbol{v} = (0111) + (1100) = (1011)$, $d(\boldsymbol{u}, \boldsymbol{v}) = w(\boldsymbol{u} + \boldsymbol{v}) = 3$.

Minimální kódová vzdálenost d_{min}. Nejmenší číslo z kódových vzdáleností všech dvojic kódových vektorů se označuje jako *minimální kódová vzdálenost* d_{min} kódu *K*. Pro každý lineární kód pak platí, že minimální kódová vzdálenost d_{min} lineárního kódu je rovna nejmenší Hammingově váze ze všech nenulových slov [1].

Věta 3.2.1. Lineární kód (n,k) má minimální kódovou vzdálenost d_{min}, pokud každá skupina $d_{min} - 1$ sloupců jeho kontrolní matice je množinou lineárně nezávislých vektorů a současně existuje alespoň jedna skupina d_{min} sloupců kontrolní matice, které jsou lineárně závislé. [1, s. 132]

3.3 Vlastnosti kódu

Mezi základní žádané vlastnosti kódu patří jeho schopnost detekovat popřípadě opravit chyby do určité násobnosti. V této kapitole proto bude definováno související názvosloví a vysvětlen způsob, jakým kódy detekují a opravují chyby. V této kapitole je také odvozeno, jaká musí být hodnota parametru minimální kódové vzdálenosti d_{min}, aby kód splňoval zadané kritérium pro výběr: tedy schopnost kódu opravit jednonásobnou a zároveň detekovat dvojnásobnou chybu.

Chybový vektor. Pokud dojde při přenosu nebo zpracování zakódované informace k chybě, změní se kódový vektor $\boldsymbol{v}$ na vektor $\boldsymbol{w} = \boldsymbol{v} + \boldsymbol{e}$, kde $\boldsymbol{e}$ je nenulový *chybový vektor*. Hammingova váha vektoru $\boldsymbol{e}$ se nazývá *váha chyby* [1].

Násobnost chyb. Chyba je označována jako t-násobná, pokud váha chyby je rovna t. Pokud se např. vyslané slovo $\boldsymbol{v} = (1001)$ změnilo vlivem poruchy na slovo $\boldsymbol{w} = (1101)$, pak chybový vektor má tvar $\boldsymbol{e} = (0100)$ a jeho váha $w(\boldsymbol{e}) = 1$. Takovou chybu označujeme jako *jednonásobnou* – chyba nastala pouze v jednom bitu přenášené informace. Analogicky chyba s vahou 2 se nazývá *dvojnásobná* [1].

Detekce chyb. Pokud bylo vysláno kódové slovo a přijato slovo, které není kódové, pak jsme *objevili* (*detekovali*) *chybu*. Pokud jsme přijali kódové slovo, pak buď nedošlo k chybě, nebo jsme chybu neobjevili, protože vyslané kódové slovo mohlo být změněno na jiné kódové slovo. Nejvyšší násobnost chyb, které jsme schopni v kódu detekovat je menší než minimální kódová vzdálenost [1].

Syndrom. K objevování chyb lineárního kódu délky n můžeme využít kontrolní matici $\boldsymbol{H}$. Pak pro každé slovo $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$ definujeme slovo a $\boldsymbol{s} = (s_1, s_2, \ldots, s_m)$ předpisem:

$$\boldsymbol{H} \cdot \boldsymbol{v}^{\boldsymbol{T}} = \boldsymbol{H} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_m \end{bmatrix} \tag{3.7}$$

kde $\boldsymbol{H}$ je kontrolní matice typu [m x n] a $\boldsymbol{v}^{\boldsymbol{T}}$ značí transponovaný vektor $\boldsymbol{v}$. Slovo $\boldsymbol{s}$ se nazývá *syndromem slova* $\boldsymbol{v}$. Je-li syndrom nenulový, víme, že došlo k chybě, neboť pro každé kódové slovo platí rovnice (3.2)

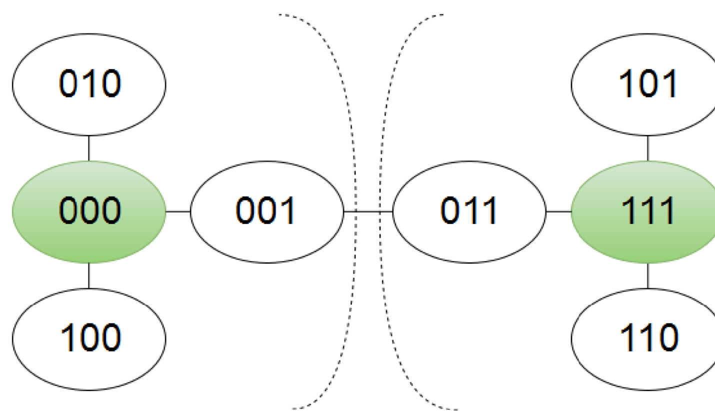
Věta 3.3.1. Přijaté slovo má stejný syndrom jako chybové slovo. Důkaz tohoto důležitého tvrzení můžeme najít např. v [2, s. 58].

Oprava chyb. *Oprava* nezávislých *chyb* v kódovém vektoru vychází z tzv. *principu maximální věrohodnosti*, podle nějž je pravděpodobnost správné opravy největší, pokud přijaté nekódové slovo opravíme na takové kódové slovo, které má od přijatého slova nejmenší Hammingovu vzdálenost [1].

Má-li kód minimální vzdálenost $d_{min} = 2t + 1$, pak množina nekódových slov, ležících ve vzdálenosti $t + 1$ kolem každého kódového slova má s ostatními množinami okolo ostatních kódových slov prázdný průnik. Proto takový kód může buď detekovat všechny nezávislé chyby až do násobnosti $2t$, nebo opravit všechny t-násobné chyby.

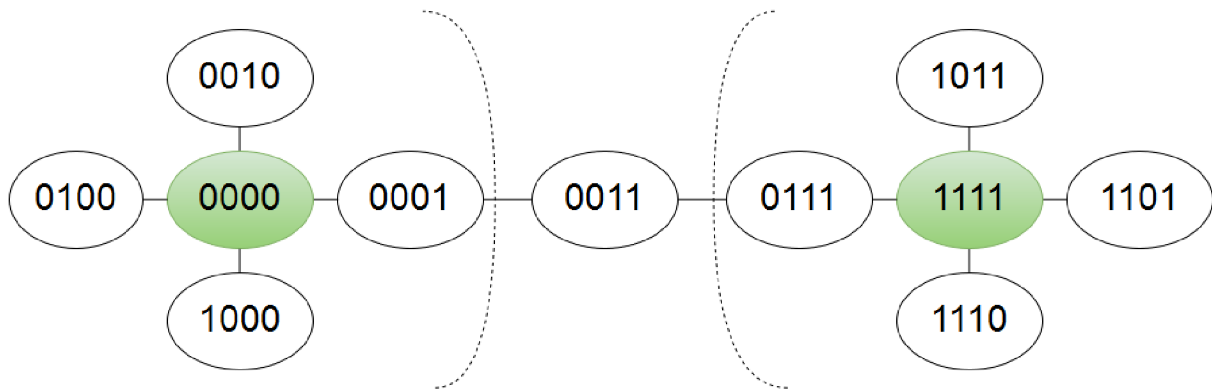
Má-li kód minimální vzdálenost $d_{min} = s + t + 1$, kde $s > t$, pak může opravit až t-násobné chyby a současně detekovat všechny chyby až do násobnosti s [2].

Příklad: Kódem s minimální vzdáleností $d_{min} = 2t + 1 = 3 \mid t = 1$, je např. binární opakovací kód délky $n = 3$, kde se každý informační znak n-krát opakuje. Tento kód má dvě kódová slova K ∈ {000, 111}. Při přenosu kódového slova $\boldsymbol{v}$ = (000) nastane dvojnásobná chyba, tj. chyba násobnosti větší než t, tak, že vznikne slovo $\boldsymbol{w}$ = (011). Protože jsme přijali nekódové slovo, můžeme říci, že jsme detekovali chybu. Podle principu maximální věrohodnosti by však toto slovo bylo nesprávně opraveno na kódové slovo $\boldsymbol{u}$ = (111).



Obr. 2: Kód s minimální Hammingovou vzdáleností $d_{min} = 3$. [6]

Aby tento kód dokázal rozlišit jednonásobnou a dvojnásobnou chybu, je nutné, aby jeho kódová vzdálenost $d_{min} \geq 4$. Kód s minimální vzdáleností $d_{min} = 4$ je např. binární opakovací kód délky $n = 4$, K ∈ {0000, 1111}. Pokud při přenosu kódového slova $\boldsymbol{v}$ = (0000) nastane dvojnásobná chyba a vznikne slovo $\boldsymbol{w}$ = (0011), nelze jednoznačně určit původní kódové slovo, neboť obě kódová slova mají od přijatého slova stejnou Hammingovu vzdálenost. Detekovali jsme tedy dvojnásobnou *neopravitelnou chybu*.



Obr. 3: Kód s minimální Hammingovou vzdáleností d_{min} = 4. [6]

Věta 3.3.2. Pokud požadujeme možnost opravit jednonásobnou chybu (tedy $t = 1$) a zároveň detekovat dvojnásobnou chybu (tedy $s = 2$), pak minimální vzdálenost tohoto kódu musí být alespoň $d_{min} = s + t + 1 = 4$.

3.4 Zástupci lineárních kódů

Mezi nejjednodušší a nejvýznamnější grupové kódy umožňující opravu jednonásobných chyb patří Hammingův kód. Hammingovy kódy mají snadný způsob dekódovaní a jsou *perfektní*, tj. mají nejmenší možnou redundanci.

Hammingův kód. *Hammingův kód* je grupový kód (n,k) s minimální vzdáleností $d_{min} = 3$, který je zadán kontrolní maticí $\boldsymbol{H}$, jejíž sloupce tvoří všechny nenulové $(n - k)$-bitové vektory a to každý právě jednou. Toto tvrzení vychází z věty 3.2.1. Na uspořádání sloupců kontrolní matice nezáleží [1].

- **Minimální kódová vzdálenost.** Minimální vzdálenost Hammingova kódu je [2]:

$$d_{min} = 3$$

- **Délka kódu.** Hammingův kód s $n - k = r$ kontrolními znaky má délku:

$$n = 2^r - 1$$

  To je zřejmé z definice Hammingova kódu, neboť větší počet $(n - k)$-bitových sloupců kontrolní matice $\boldsymbol{H}$ není možné sestavit [1].

- **Informační poměr.** Informační poměr Hammingova kódu roste velmi rychle k 1 s rostoucí délkou kódu. Obecně je dán rovnicí:

$$R = \frac{k}{n} = \frac{2^r - 1 - r}{2^r - 1}$$

  Kde R značí informační poměr, n značí celkovou délku kódu, k značí počet informačních bitů a $r = n - k$ značí počet kontrolních bitů [2].

Na následujícím příkladu je ukázán Hammingův kód určený kontrolní maticí $\boldsymbol{H}$, jejíž sloupce mají takové uspořádání, aby tento kód byl systematický. Na tomto kódu je ukázán způsob kódování a dekódování.

Příklad. Je dán Hammingův kód (7,4) který je určen kontrolní maticí:

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & | & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & | & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & | & 0 & 0 & 1 \end{bmatrix}$$

Jedná se systematický kód, neboť jeho kontrolní matice $\boldsymbol{H}$ je typu (3.5). Jeho generující matici $\boldsymbol{G}$ můžeme získat podle (3.4) a bude mít tvar:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & | & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & | & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & | & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & | & 0 & 1 & 1 \end{bmatrix}$$

Tento kód má délku $n = 7$, kde jeho prvních $k = 4$ bitů tvoří informační bity (vzájemně nezávislé, volitelné), které mohou nabývat všech binárních kombinací tj. $2^k = 16$ kombinací. Tyto jsou následovány $k = 3$ kontrolními bity (závislými).

Při kódování slov vznikne kódový vektor $\boldsymbol{v} = (v_0, v_1, \ldots, v_6)$ z informačního vektoru $\boldsymbol{v} = (v_0, v_1, v_2, v_3)$ jeho součinem s generující maticí $\boldsymbol{G}$, tj. $(v_0, v_1, v_2, v_3) \cdot \boldsymbol{G} = (v_0, v_1, v_2, v_3, v_0 + v_1 + v_2, v_0 + v_1 + v_3, v_0 + v_2 + v_3)$. Pro kontrolní bity tak můžeme odvodit rovnice:

$$\begin{aligned} v_4 &= v_0 + v_1 + v_2 \\ v_5 &= v_0 + v_1 + v_3 \\ v_6 &= v_0 + v_2 + v_3 \end{aligned} \tag{3.8}$$

Druhý způsob kódování slov vychází z vlastností kontrolní matice. Pro kódový vektor $\boldsymbol{v} = (v_0, v_1, \ldots, v_6)$ musí platit rovnice (3.2), tedy že jeho součin s kontrolní maticí je roven nulovému vektoru, což po rozepsání jednotlivých skalárních součinů dává rovnice:

$$\begin{aligned} v_0 + v_1 + v_2 \quad\quad + v_4 \quad\quad\quad &= 0 \\ v_0 + v_1 \quad\quad + v_3 \quad\quad + v_5 \quad &= 0 \\ v_0 \quad\quad + v_2 + v_3 \quad\quad\quad\quad + v_6 &= 0 \end{aligned}$$

Pro jednotlivé kontrolní bity tak opět získáme rovnice (3.8). Pokud při přenosu došlo k chybě, byl přijat vektor $\boldsymbol{w} = \boldsymbol{v} + \boldsymbol{e}$, kde $\boldsymbol{e} = (e_0, e_1, \ldots, e_6)$ je chybový vektor. Při dekódování můžeme naproti tomu využít skalární součin přijatého vektoru $\boldsymbol{w} = (w_0, w_1, \ldots, w_6)$ s kontrolní maticí $\boldsymbol{H}$ viz (3.7). Výsledkem součinu je syndrom $\boldsymbol{s}$, který je stejný, jako syndrom chybového vektoru $\boldsymbol{e}$, neboť:

$$\boldsymbol{s} = \boldsymbol{H} \cdot \boldsymbol{w}^T = \boldsymbol{H} \cdot (\boldsymbol{v} + \boldsymbol{e})^T = \boldsymbol{H} \cdot \boldsymbol{v}^T + \boldsymbol{H} \cdot \boldsymbol{e}^T = \boldsymbol{0} + \boldsymbol{H} \cdot \boldsymbol{e}^T$$

Pokud je tedy $\boldsymbol{s} = \mathbf{0}$, pak buď k chybě nedošlo, nebo má $\boldsymbol{e}$ tvar kódového slova, takže nastala chyba nejméně v $d_{min} = 3$ bitech. Taková chyba se nazývá *nedetekovatelná* [1].

Pokud je $\boldsymbol{s} \neq \mathbf{0}$, pak předpokládáme chybu v i-tém bitu chybového vektoru $\boldsymbol{e} = (e_0, \ldots, e_i, \ldots, e_6)$. Syndrom $\boldsymbol{s}$ je pak roven i-tému sloupci kontrolní matice. Syndromu přijatého slova tak může být převeden na chybový vektor s a znegováním chybného bitu může být přijatý vektor opraven [1]

Perfektní kód. Takový kód, který má při daném počtu informačních bitů nejmenší myslitelnou redundanci, se nazývá *perfektní kód.*

Hammingův kód je perfektní kód pro opravu jednonásobných chyb. Pokud však při přenosu dojde k dvojnásobné chybě, vzniklé slovo má stejný syndrom, jako některá jednonásobná chyba a tak dojde k nesprávné opravě. Tento kód proto nevyhovuje kritériu, aby kód byl schopen opravit jednonásobnou a zároveň detekovat dvojnásobnou chybu. Jak bylo řečeno ve větě 3.3.2, aby kód splňoval toto kritérium, musí mít minimální vzdálenost $d_{min} = s + t + 1 = 4$. Takovým kódem je *rozšířený Hammingův kód.*

Rozšířený Hammingův kód. Pokud provedeme konstrukci *rozšíření* Hammingova kódu, tj. ke každému jeho kódovému slovu přidáme jeden kontrolní bit tak, aby kódové slovo mělo sudou paritu, vznikne *rozšířený Hammingův kód.* [2]

- **Minimální kódová vzdálenost.** Pokud má binární kód K minimální kódovou vzdálenost d, pak rozšířený kód má minimální vzdálenost $d + 1$, pokud je d liché číslo a vzdálenost d, pokud je d sudé číslo. Rozšířením Hammingova kódu tak získáme kód s minimální vzdáleností [2]:

$$d_{min} = 4$$

- **Délka kódu.** Rozšířený Hammingův kód (n,k) má délku:

$$n = 2^{r-1}$$

kde $r = n - k$ značí počet kontrolních bitů a počet informačních bitů $k = 2^{r-1} - \mathrm{r}$ [1].

- **Informační poměr.** Informační poměr pak můžeme snadno odvodit z předchozích rovnic.

$$R = \frac{k}{n} = \frac{2^{r-1} - r}{2^{r-1}} = 1 - \frac{r}{2^{r-1}}$$

Doplnění kódových slov na sudou paritu se projeví v kontrolní matici $\boldsymbol{H}$ doplněním všech řádků o nulu na konci a přidáním řádku samých jedniček. Díky tomu budou mít vždy syndromy všech dvojnásobných chyb odlišné od syndromů chyb jednonásobných [2].

V pamětech se zpravidla používají slova takové délky, z níž počet informačních bitů je mocninou čísla dva, tedy např. k = 8, 16, 32, 64, …. Délku kódu proto možné zkrátit vynecháním nepotřebných informačních bitů. Např. rozšířený Hammingův kód (32,26) lze upravit na kód (22,16) vynecháním nepotřebných 10 informačních bitů. Počet kontrolních bitů obou kódu však zůstává stejný [1].

4 CYKLICKÉ KÓDY

4.1 Důležité pojmy

Řadu vlastností cyklických kódů lze nejsnáze pochopit, pokud kódová slova budeme reprezentovat pomocí polynomů. Každý kódový vektor binárního cyklického kódu (n,k) se souřadnicemi z $GF(2)$ můžeme reprezentovat jako polynom s koeficienty z $GF(2)$ stupně nižšího než n. Těmto polynomům říkáme kódové polynomy. Vektor $\boldsymbol{v} = (v_{n-1}, v_n, \ldots, v_0)$ tedy budeme zapisovat jako polynom:

$$x^{n-1}v_{n-1} + x^{n-2}v_{n-2} + \cdots + xv_1 + v_0 \quad (4.1)$$

Kde n je délka slova a + značí operaci sčítání modulo 2 [1].

Cyklický kód. Lineární kód, který má tu vlastnost, že je-li $\boldsymbol{v} = (v_{n-1}, v_n, \ldots, v_0)$ kódovým vektorem, je kódovým vektorem i vektor $\boldsymbol{v}' = (v_{n-2}, v_{n-3}, \ldots, v_{n-1})$, označujeme jako *cyklický kód*. Této vlastnosti říkáme *cyklická vlastnost*, popřípadě říkáme, že cyklické kódy splňují *podmínku cykličnosti* [1].

Příklad. Slovo 00101 binárního kódu délky $n = 5$ můžeme zapsat jako kódový polynom $x^2 + 1$, neboli $0{\cdot}x^4 + 0{\cdot}x^3 + 1{\cdot}x^2 + 0{\cdot}x^1 + 1{\cdot}x^0$Cyklickým posunem získáme slovo 01010 tedy $x^3 + x = x(x^2 + 1)$. Dalším cyklickým posuvem bychom získali slovo 10100 tedy $x^4 + x^2 = x^2(x^2 + 1)$.

Okruh $T^{(n)}$. Pro cyklické kódy je důležitý okruh polynomů $T^{(n)} = T/(x^n - 1)$ jehož prvky jsou všechny polynomy stupně nižšího, než n. Tyto polynomy reprezentují slova délky n, neboť každé slovo můžeme dle (4.1) reprezentovat jako polynom [2].

Cyklický kód je pak *ideálem* okruhu $T^{(n)}$, což znamená, že pro libovolný kódový polynom $v(x)$ jsou i všechny násobky $q(x)\ v(x)$ kde $q(x) \in T^{(n)}$, kódovými polynomy [2, s. 105].

Cyklický posun. Cyklický posun je realizován vynásobením polynomu $v(x)$ proměnnou x s tím, že $x^n = 1$. Proto chápeme binární cyklický kód K jako lineární podprostor okruhu $T^{(n)} = T/(x^n - 1)$, tedy jako vektorový prostor, jehož prvky tvoří všechny zbytkové třídy polynomů nad konečným tělesem $GF(2)$ modulo $(x^n - 1)$ [2].

- Sčítání zbytkových tříd se provádí prostřednictvím sčítání koeficientů u stejných mocnin x, až na to, že koeficienty se sčítají podle pravidel platných v $GF(2)$.
- Násobení se provádí stejně jako běžné násobení polynomů nad tělesem reálných čísel s tím, že pokud by součin byl stupně n nebo vyššího, provede se ještě operace modulo $(x^n - 1)$ [1].

Příklad. Binární kód celkové parity délky $n = 4$ je cyklický kód. Jeho prvky jsou tedy všechny zbytkové třídy všech polynomů s koeficienty z $GF(2)$ modulo $(x^4 - 1)$:

0000	=	0	=	$0 \cdot (x + 1)$
0011	=	$x + 1$	=	$1 \cdot (x + 1)$
0101	=	$x^2 + 1$	=	$(x + 1) \cdot (x + 1)$
1001	=	$x^3 + 1$	=	$(x^2 + x + 1) \cdot (x + 1)$
0110	=	$x^2 + x$	=	$x \cdot (x + 1)$
1010	=	$x^3 + x$	=	$(x^2 + x) \cdot (x + 1)$
1100	=	$x^3 + x^2$	=	$x^2 \cdot (x + 1)$
1111	=	$x^3 + x^2 + x + 1$	=	$(x^2 + 1) \cdot (x + 1)$

Jako příklad počítání s kódovými slovy, které tvoří zbytkové třídy polynomů nad konečným tělesem $GF(2)$ modulo $(x^n - 1)$, uvedeme součin $\{x^3 + x^2\}\{x^2 + x\} = \{x^5 + x^4 + x^4 + x^3\}$. Vzhledem k pravidlům sčítání nad $GF(2)$ je výsledkem polynom $\{x^5 + x^3\}$. Jelikož tento polynom je stupně vyššího než n, provedeme operaci modulo $(x^4 - 1)$ jejíž výsledkem je polynom $\{x^3 + x\}$ [1].

V tomto kódu je nenulový kódový polynom nejnižšího stupně polynom $x + 1$. Ostatní kódové polynomy jsou právě všechny násobky polynomu $x + 1$ v okruhu ${}^4\mathbb{Z}_2$. Polynom s těmito vlastnosti se nazývá generující polynom [1].

Generující polynom. Každý netriviální cyklický (n,k)-kód K obsahuje kódové slovo $g(x)$, které jako polynom má stupeň $n - k$. Všechna další kódová slova jsou násobky polynomu $g(x)$.

Věta 4.1.1. Generující polynom má tyto vlastnosti [2].

- Kód K sestává ze všech násobků polynomu $g(x)$ v $T^{(n)}$, tj.

  $K = \{q(x)g(x) \mid q(x) \in T^{(n)}\}$;

- Polynomy $g(x)$, $x\,g(x)$, …, $x^{n-1}g(x)$ tvoří bázi kódu K
- Polynom $g(x)$ dělí polynom $x^n - 1$ beze zbytku.

Kontrolní polynom. Každý cyklický kód může být také určen svým *kontrolním polynomem* $h(x)$. Ten je definován jako:

$$h(x) = (x^n - 1) : g(x) \tag{4.2}$$

Cyklický kód s kontrolním polynomem $h(x)$ sestává právě ze všech polynomů $v(x)$ v okruhu $T^{(n)}$, pro které platí:

$$v(x)\,h(x) = 0$$

Tento polynom tak nahrazuje funkci kontrolní matice [2].

Významný způsob určení cyklického kódu je stanovení společných kořenů všech polynomů tohoto kódu. Tyto kořeny lze najít v určitém rozšíření kódové abecedy. Z těchto kořenů je totiž vytvořit generující polynom a kontrolní matici. [2]

Kořen polynomu. Kořenem polynomu $f(x)$ nad tělesem T se nazývá takový prvek $a \in T$, pro který platí $f(a) = 0$ [2].

Příklad. Polynom $x^2 + 1$ má v tělese $\mathbb{Z}_2$ kořen $a = 1$, neboť podle pravidel platných v $\mathbb{Z}_2$ dostaneme po dosazení $1^2 + 1 = 1 + 1 = 0$. V tělese $\mathbb{Z}_3$ tento polynom žádný kořen nemá.

Kořenový činitel. Polynom $f(x)$ má kořen a právě, když je dělitelný tzv. *kořenovým činitelem* $(x - a)$. Libovolný polynom tedy můžu rozložit na součin jeho kořenových činitelů. Polynom stupně n má nejvýše n kořenů a je dělitelný polynomem $(x - a_1)\ (x - a_2)\ \dots\ (x - a_k)$ kde $k \leq n$ [2].

Pro cyklické kódy je důležitý rozklad polynomu $(x^n - 1)$, neboť, jak bylo řečeno ve větě 4.1.1, generující polynom cyklického kódu je členem rozkladu tohoto polynomu.

Charakteristika tělesa. *Charakteristikou tělesa T* se nazývá nejmenší číslo p takové, že součet $1 + 1 + \dots + 1$ (s p sčítanci) je roven 0. Např. těleso $\mathbb{Z}_2$ má charakteristiku 2 a každé těleso T se stejnou charakteristikou je rozšířením tělesa $\mathbb{Z}_2$ tj. těleso $GF(2^n) = \mathbb{Z}_2/q(x)$. Prvky 0 a 1 tělesa T tvoří těleso $\mathbb{Z}_2$ [2].

Věta 4.1.2. Buď T těleso charakteristiky 2. Jestliže polynom $f(x)$ nad tělesem $\mathbb{Z}_2$ má kořen a v tělese T, potom jsou i a^2, a^4, a^6 … kořeny polynomu $f(x)$. [2]

Řád. Prvek $a \neq 0$ má v tělese řád r, jestliže je r nejmenší číslo 1,2,3, … takové, že $a^r = 1$. Každý prvek konečného tělesa má konečný řád. Prvky $1, a, a^2, \dots a^{r-1}$ jsou navzájem různé a platí $a^n = 1$, právě když n je násobkem čísla r (n = 1, 2, 3) [2].

Rozklad polynomu $(x^n - 1)$. Rozklad tohoto polynomu je snadný v rozšířeném tělese T, které obsahuje prvek α řádu n. V tomto tělese totiž platí:

$$x^n - 1 = (x - 1)\ (x - \alpha)\ (x - \alpha^2)\ \dots\ (x - \alpha^{n-1}) \tag{4.3}$$

Přitom mocniny $1, \alpha, \dots \alpha^{n-1}$ jsou navzájem různé.

Těleso T najdeme jako $T = GF(2^m)$, kde m je nejmenší číslo takové, že n je dělitelem čísla $2^m - 1$. Takové těleso je ale možné najít pouze tehdy, pokud je 2 dělitelem čísla n. Důkaz tohoto tvrzení je možné najít např. v literatuře [2 s.125]

Věta 4.1.3. Délka n každého binárního cyklického kódu, který je možné určit pomocí generujících kořenů, musí být liché číslo [2].

Generující kořeny. Každý binární cyklický kód K délky n je charakterizován svým generujícím polynomem $g(x)$, který je členem rozkladu polynomu $x^2 - 1$. V tělese T má tedy i polynom $g(x)$ rozklad [2]:

(4.4) $$g(x) = (x - \alpha^{i_1})(x - \alpha^{i_2}) \dots (x - \alpha^{i_s})$$

Víme, že každý kódový polynom je násobek generujícího polynomu. Proto polynom $v(x)$ je kódový právě tehdy, pokud má *generující kořeny* $\alpha^{i1}, \alpha^{i2}, \dots \alpha^{is}$. Stručněji:

$$v(x) \in K \leftrightarrow v(\alpha^{i_k}) = 0 \mid k = 1, 2, \dots s$$

Kód K je tedy úplně charakterizován svými generujícími kořeny:

$$\alpha^{i_1}, \alpha^{i_2}, \dots, \alpha^{i_s}$$

Příklad. Naší snahou je najít binární cyklický kód délky $n = 7$. Proto hledáme těleso $GF(2^m)$, které je rozšířením tělesa $\mathbb{Z}_2$. Nejmenší číslo m takové, aby n bylo dělitelem čísla $2^m - 1$ je číslo 3, neboť $2^3 - 1 = 7$ a $7/7 = 1$. V tělese $GF(2^3) = GF(8)$ zvolíme jako α prvek x, který má řád 7 a proto platí

$$x^7 - 1 = (x - 1)(x - \alpha)(x - \alpha^2) \dots (x - \alpha^6)$$

Např. Hammingův (7,4)-kód je kód délky $n = 7$, má generující polynom $g(x) = x^3 + x + 1$ (jediný kódový polynom stupně $n - k = 3$). Prvek α (= x) v tělese $GF(8)$ je kořenem polynomu $g(x)$. Odtud dle věty 4.1.2 plyne, že i α^2 a α^4 jsou také jeho kořeny a tedy:

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4)$$

Proto můžeme říci, že α, α^2 a α^4 jsou generující kořeny Hammingova kódu. Sám kořen α je však také generující kořen, neboť pro každé kódové slovo platí $v(\alpha) = 0$ [2].

Kontrolní matice cyklického kódu. Binární cyklický kód délky n s generujícími kořeny $\alpha^{i1}, \alpha^{i2}, \dots \alpha^{is}$ má tuto kontrolní matici:

(4.5) $$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha^{i_1} & (\alpha^{i_1})^2 & (\alpha^{i_1})^3 & \cdots & (\alpha^{i_1})^{n-1} \\ 1 & \alpha^{i_2} & (\alpha^{i_2})^2 & (\alpha^{i_2})^3 & \cdots & (\alpha^{i_2})^{n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \alpha^{i_s} & (\alpha^{i_s})^2 & (\alpha^{i_s})^3 & \cdots & (\alpha^{i_s})^{n-1} \end{bmatrix}$$

V rozšíření $T = GF(2^m)$ tělesa $\mathbb{Z}_2$ jsou prvky tohoto tělesa polynomy $a(x)$ stupně $< m$, které můžeme chápat jako sloupce jejich koeficientů:

$$\begin{bmatrix} a_{m-1} \\ a_{m-2} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix}$$

V tomto smyslu matice o k řádcích nad tělesem T se stává maticí o km řádcích [2].

Příklad. Cyklický Hammingův kód má jeden generující kořen $\alpha = x$ v tělese $GF(8)$. Proto jeho kontrolní matici můžeme zapsat ve formě [2]:

$$[1 \quad \alpha \quad \alpha^1 \quad \alpha^2 \quad \alpha^3 \quad \alpha^4 \quad \alpha^5 \quad \alpha^6]$$

Přitom generující kořen α je prvek $x = 0 \cdot x^2 + 1 \cdot x^1 + 0 \cdot x^0$ který můžeme zapsat jako sloupec:

$$\alpha = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

A např. α^3, což je prvek $x + 1$, můžeme zapsat jako:

$$\alpha^3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Kontrolní matici Hammingova kódu tak můžeme přepsat do binárního tvaru:

$$\boldsymbol{H} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{4.6}$$

4.2 Vlastnosti cyklických kódů

Systematické kódování. Každý cyklický kód je podle věty 3.1.1 ekvivalentní se systematickým kódem V případě cyklických kódů je toho možné docílit tak, že každý kódový polynom zapíšeme od nejvyšší mocniny k nejnižší. Pro cyklický systematický kód existuje způsob kódování, označovaný jako *systematické kódování*, který je založený na dělení informačních znaků generujícím polynomem. Pokud je totiž z informačních znaků $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$ vytvořen polynom [2]:

$$u(x) = u_0 x^{n-1} + u_1 x^{n-2} + \cdots + u_{k-1} x^{n-k}$$

Pak platí:

$$u(x) = q(x)g(x) + r(x), \qquad \text{kde } q(x) \in T^{(n)}$$

kde $g(x)$ značí generující polynom a $r(x)$, jehož stupeň je $st < n - k$, značí zbytek po dělení. Proto odečtením zbytku po dělení $r(x)$ od původního polynomu $u(x)$ vnikne kódové slovo:

$$q(x)g(x) = u(x) - r(x)$$

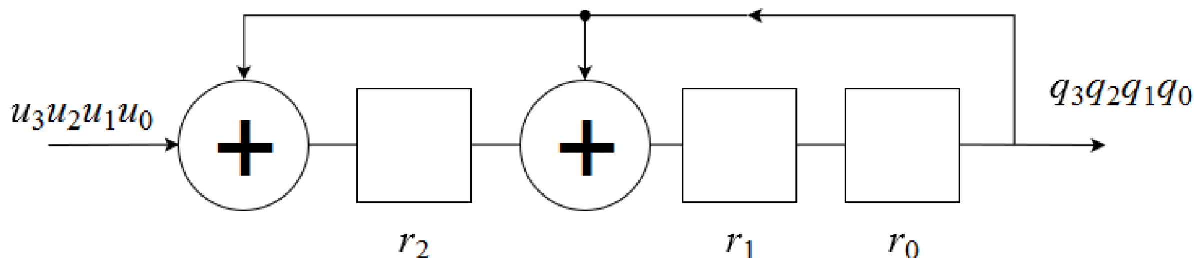
Pokud označíme:

$$-r(x) = r_k x^{n-k-1} + r_{k+1} x^{n-k-2} + \cdots + r_{n-1} x + r_n$$

Pak bude vysláno kódové slovo:

$$u_0 u_1 \ldots u_{k-1} r_k \ldots r_n$$

Binární cyklické kódy mají snadný způsob realizace kódování informačních znaků. To proto, že násobení i dělení binárním polynomem je snadné vytvořit číslicovými obvody. Např. pro systematické kódování Hammingova (7,4)-kódu dělíme polynomem $g(x) = x^3 + x + 1$. Tato operace může být zajištěna číslicovým obvodem s dvěma binárními sčítačkami a třemi posuvnými registry [2]:



Obr. 4: Realizace dělení polynomem. [2]

Do obvodu vstupují koeficienty polynomu $u(x)$ a vystupují koeficienty podílu $q(x) = q_0x^3 + q_1x^2 + q_2x + q_3$ a po vystoupení posledního z nich v registrech zůstávají koeficienty zbytku $r(x) = r_0x^2 + r_1x + r_2$ [2].

V předchozích úvahách jsme se zabývali pouze chyb, které se v přenášené zprávě objevují jen náhodně. V určitých případech se však vyskytují chyby ve shlucích, tj. v blízkosti chybného bitu lze očekávat další chybný bit. Typickým příkladem vzniku shluku chyb je např. porušení magnetického disku. Mezi důležitou vlastnost každého cyklického kódu patří schopnost detekovat a opravit shlukové chyby do určité délky [2].

Shluk chyb. *Shlukem chyb* délky b se nazývá slovo $\boldsymbol{e} = e_1e_2 \ldots e_n$, jehož všechny nenulové složky tvoří část, ležící mezi b po sobě následujícími znaky. Tedy slovo tvaru [2]:

$$\boldsymbol{e} = 00 \ldots 0e_ie_{i+1} \ldots e_{i+b-1}0 \ldots 00.$$

To znamená, že v přenášené zprávě na prvním a posledním místě části přijatého slova ovlivněného shlukem chyb budou chybné bity, zatímco každý z mezilehlých bitů může a nemusí být chybný. [1]

Detekování shluku chyb. Každý cyklický kód s generujícím polynomem stupně b detekuje každý jednonásobný shluk chyb délky b nebo kratší. [1]

Oprava shluku chyb. K opravě shluku chyb délky b je potřebný takový cyklický kód, který dokáže detekovat každou chybu ve tvaru dvojice shluků chyb, kde každý ze shluků má nejvýše délku b. Taková dvojice shluků totiž potom není kódovým slovem, každý ze shluků musí mít proto nutně jiný syndrom a všechny b-bitové shluky jsou pak již syndromem rozlišitelné a opravitelné [1].

4.3 Zástupci cyklických kódů

Cyklický Hammingův kód. Hammingův (n,k)-kód je zadán kontrolní maticí $\boldsymbol{H}$, jejíž sloupce tvoří všechny nenulové $(n-k)$-bitové vektory a to každý právě jednou. Na uspořádání sloupců kontrolní matice nezáleží, pouze při vhodném uspořádáním však vznikne *cyklický Hammingův kód.* Nalezení správného uspořádání sloupců kontrolní matice cyklického

Hammingova kódu bylo ukázáno v (4.6). Tento kód má stejné parametry jako obecný Hammingův kód, proto je zde jen stručně shrneme:

- **Minimální kódová vzdálenost.**

$$d_{min} = 3$$

- **Délka kódu.**

$$n = 2^r - 1$$

- **Informační poměr.**

$$R = \frac{k}{n} = \frac{2^r - 1 - r}{2^r - 1}$$

Protože cyklický Hammingův kód má minimální kódovou vzdálenost $d_{min} = 3$, nedokáže dle věty 3.3.2 zároveň detekovat dvojnásobnou a opravit jednonásobnou chybu. Tento kód tedy nesplňuje zadané kritérium pro výběr vhodného kódu.

Binární BCH kódy. Binární BCH kód s plánovanou vzdáleností $d = 2, 3, 4, \dots$ je kód liché délky $n \geq d$ s generujícími kořeny:

$$\alpha, \alpha^2, \alpha^3 \dots, \alpha^{d-1}$$

kde α je prvek n-tého řádu v tělese $GF(2^m)$.

BCH kódy jsou jednou z nejdůležitějších tříd bezpečnostních kódů. Tuto třídu objevili R. C. Bose a D. K. Ray Chaudhuri a nezávisle na nich A. Hocquenghem – podle jejich iniciálů se nazývají právě BCH kódy. Jsou velmi dobré pro opravu velkého množství chyb, mají dobrý vztah mezi počtem informačních znaků a počtem opravovaných chyb, výhodou je také velká volitelnost parametrů a také dobře vypracované dekódovací metody [4].

- **Plánovaná vzdálenost *d*.** Sudé mocniny mezi generujícími kořeny jsou zbytečné, neboť každý polynom $v(x) \in T^{(n)}$ má s každým kořenem α^i také kořeny α^{2i}, α^{4i}, …. Proto BCH kódy s *plánovanou vzdáleností* $d = 2t$ nebo $d = 2t + 1$ jsou totožné a mají generující kořeny [2]:

  $$\alpha, \alpha^3, \alpha^5 \dots, \alpha^{2t-1}$$

  Proto budeme vždy předpokládat, že plánovaná vzdálenost d je vždy liché číslo [2]. Lze dokázat [2, s. 148], že kód s plánovanou vzdáleností d má minimální vzdálenost $d_{min} \geq d$. Kromě toho, jsou-li po sobě jdoucí mocniny α^0, α^1, α^2, ... α^{d-2} z $GF(2^m)$ kořeny všech kódových polynomů, pak $d_{min} \geq d$ [1].

Poznámka. Kód s plánovanou vzdáleností d může mít i větší skutečnou minimální vzdálenost a je otevřeným problémem, jak obecně určit minimální vzdálenost BCH kódu. Stejně tak je otevřený problém, určení počtu informačních a kontrolních znaků BCH kódu. Obecný vzorec můžeme najít pro kódy délky $n = 2^m - 1$, pokud plánovaná vzdálenost nepřekročí číslo $d = 2 + \sqrt{(n + 1)}$ [2].

- **Počet kontrolních znaků.** Počet kontrolních znaků binárního BCH kódu délky $n = 2^m - 1$ s plánovanou vzdáleností $d \leq 2 + \sqrt{(n + 1)}$ je [2]:

$$r = n - k = \frac{1}{2}m(d - 1)$$

- **Kontrolní matice.** BCH kód délky n s plánovanou vzdáleností d lze definovat jeho kontrolní maticí, která má $d - 2$ řádků a n sloupců [2]:

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & \alpha^{d-2} & (\alpha^{d-2})^2 & (\alpha^{d-2})^3 & \cdots & (\alpha^{d-2})^{n-1} \end{bmatrix}$$

BCH kódy pro opravu jednonásobných chyb. BCH kódy jsou zobecněním perfektních Hammingových kódů – BCH kód pro jednoduché opravy je právě Hammingův cyklický kód, neboť ten má jeden generující kořen α, jeho minimální vzdálenost $d_{min} = 3$ a kontrolní matici tvoří jeden řádek (nad tělesem $GF(2^m)$) [2].

BCH kódy pro opravu dvojnásobných chyb. BCH kódy zahrnují širokou škálu kódů, naším úkolem je však vybrat takový kód, který bude mít $d_{min} \geq 4$ a zároveň bude mít co nejmenší redundanci. Takovým kódem *BCH kód pro opravu dvojnásobných chyb* s plánovanou vzdáleností $d = 5$. Je to kód liché délky $n \geq 5$ s generujícími kořeny α a α^3, kde α je prvek n-tého řádu v některém tělese $GF(2^m)$. Jeho kontrolní matice má proto tvar [2]:

$$\boldsymbol{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \cdots & \alpha^{3(n-1)} \end{bmatrix}$$

- **Minimální kódová vzdálenost.** Víme, že α a tím i α^2, α^4, … jsou kořenem všech kódových polynomů a stejně tak α^3 a tím i α^6, α^9 … jsou kořenem všech kódových polynomů. A protože tím pádem α^0, α, α^2, α^3, α^4 jsou pěticí po sobě jdoucích mocnin α, jež jsou kořeny kódových polynomů, je [1]:

$$d_{min} \geq 6$$

- **Počet kontrolních znaků.** Počet kontrolních znaků (tj. stupeň polynomu $g(x)$) je v případě délky $n = 2^m - 1$ roven

$$r = 2m \quad \text{pro } m = 3, 4, 5 \dots$$

V případě jiné délky může být počet kontrolních znaků i menší. Způsob nalezení takového kódu však může být vcelku obtížný a může být nalezen např. v [2, s. 144].

- **Informační poměr.** Informační poměr BCH kódu s délkou $n = 2^m - 1$ můžeme odvodit z předchozích rovnic.

$$R = \frac{k}{n} = \frac{n - r}{n} = 1 - \frac{r}{2^m - 1} = 1 - \frac{r}{2^{\frac{1}{2}r} - 1}$$

Pro BCH kódy s jinou délkou n platí, že informační poměr může být i větší číslo [2].

5 SROVNÁNÍ

V předchozích kapitolách byli uvedeni zástupci lineárních kódů (Hammingův kód, rozšířený Hammingův kód) a cyklických kódů (cyklický Hammingův kód, tj. BCH kód pro jednonásobné opravy, a BCH kód pro dvojnásobné opravy.) Z těchto kódů bude vybrán ten, který splňuje zadané kritérium a zároveň má nejlepší vlastnosti.

Vhodný kód musí splňovat toto kritérium:

- Schopnost detekovat dvojnásobnou a opravit jednonásobnou chybu.

Ze všech kódů, splňující toto kritérium je naším cílem vybrat takový kód, který bude mít optimální parametry a vlastnosti:

- Co nejlepší informační poměr R (hlavní podmínka)
- Schopnost detekovat a opravit shlukovou chybu

Jak již bylo řečeno, zadaná kritéria nesplňuje Hammingův kód a proto ani cyklický Hammingův kód, neboť jejich minimální vzdálenost $d_{min} < 4$ a proto nedokáží zároveň opravit jednonásobnou a detekovat dvojnásobnou chybu, jak je řečeno ve větě 3.3.2. V další části tak budou porovnány pouze rozšířený Hammingův kód a BCH kód pro opravu dvojnásobných chyb. Jejich vlastnosti a parametry uvedené v předchozích kapitolách jsou zde shrnuty do tabulky č. 1.

Tab. 1: Srovnání parametrů a vlastností rozšířeného Hammingova kódu a BCH kódu pro dvojnásobné opravy

	Rozšířený Hammingův kód	**BCH kód pro dvojnásobné opravy**
Parametry		
Minimální vzdálenost d_{min}	$d_{min} = 4$	$d_{min} \geq 5$
Délka slova n	$n = 2^{r-1}$	$n \leq 2^m - 1 = 2^{\frac{1}{2}r} - 1$
Počet informačních znaků k	$k = 2^{r-1} - r$	$k \geq n - 2m = 2^{\frac{1}{2}r} - 1 - r$
Informační poměr R	$R = \frac{k}{n} = 1 - \frac{r}{2^{r-1}}$	$R \geq \frac{k}{n} = 1 - \frac{r}{2^{\frac{1}{2}r} - 1}$
Vlastnosti		
Počet detekovatelných chyb	2	≥ 3
Počet opravitelných chyb	1	≥ 2
Oprava shlukových chyb	Ne	Ano

Pro praktickou situaci lze přitom předpokládat dvě situace. V prvním případě je známá délka slova paměti a snahou je najít takový kód, který při daném kritériu dokáže zakódovat větší počet informačních znaků. V tabulce č. 2 jsou pro zvolené délky slov, jež jsou mocninou čísla 2, uvedeny parametry obou kódů. Z uvedených parametrů je následně vynesen graf závislosti informačního poměru na délce kódu.

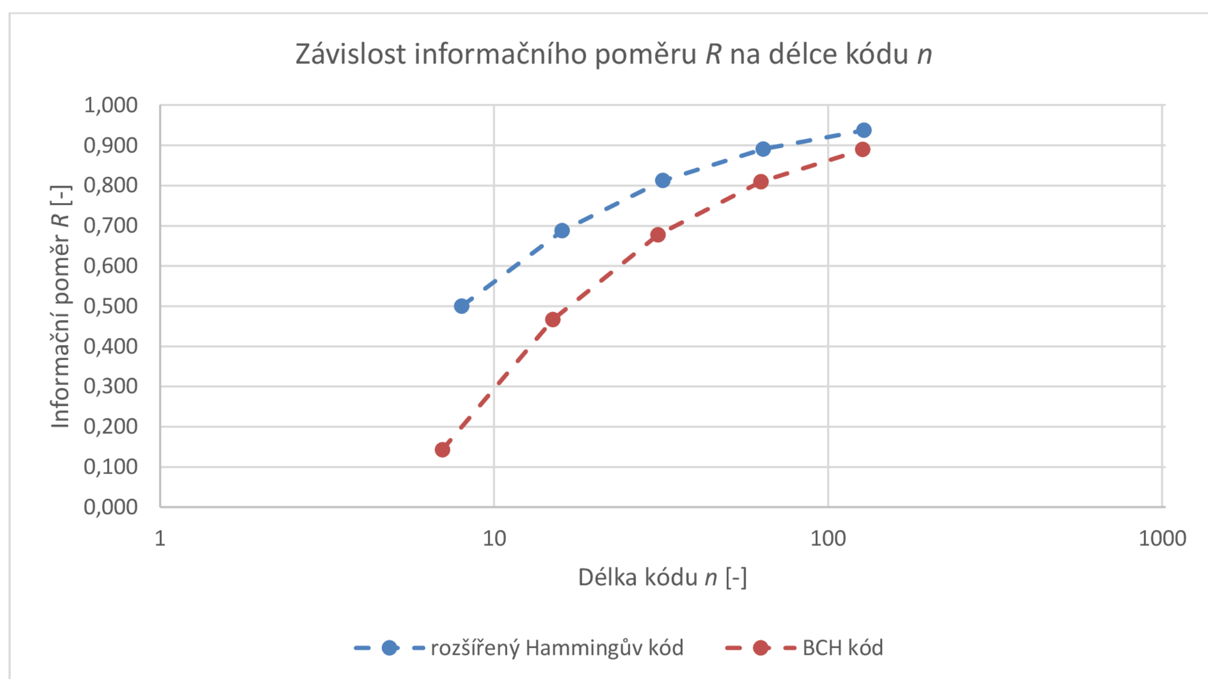
Tab. 2: Parametry srovnávaných kódů pro danou délku slov paměti

Rozšířený Hammingův kód				BCH kód pro dvojnásobné opravy				
n	***k***	***r***	***R***	***n***	***k***	***r***	***m***	***R***
8	4	4	0,500	7	1	6	3	0,143
16	11	5	0,688	15	7	8	4	0,467
32	26	6	0,813	31	21	10	5	0,677
64	57	7	0,891	63	51	12	6	0,810
128	120	8	0,938	127	113	14	7	0,890

Druhým případem je situace, kdy je znám počet informačních bitů zdrojové zprávy a je hledán takový kód, který by při daném kritériu měl co nejmenší délku kódových slov. V tabulce č. 3 jsou uvedeny parametry srovnávaných kódů pro zvolený počet informačních bitů. Protože je však obtížné určit parametry BCH kódů s délkou slova rozdílné od $n = 2^m - 1$, jsou uvedeny parametry pro takový počet informačních bitů, jaký odpovídá délce kódu $n = 2^m - 1$. Rozšířený Hammingův kód je pro tento případ zkrácen o nadbytečné informační bity.

Tab. 3: Parametry srovnávaných kódů pro daný počet informačních bitů

Rozšířený Hammingův kód zkrácený o nadbytečné informační bity				BCH kód pro dvojnásobné opravy				
k	***n***	***r***	***R***	***k***	***n***	***r***	***m***	***R***
7	12	5	0,583	7	15	8	4	0,467
21	27	6	0,778	21	31	10	5	0,677
51	58	7	0,879	51	63	12	6	0,810
113	121	8	0,934	113	127	14	7	0,890



Graf 1. Graf závislosti informačního poměru R na délce kódu k porovnávaných kódů

Ze srovnání je patrné, že z porovnávaných kódů, které splňují zadané kritérium, má optimálnější parametry rozšířený Hammingův kód, neboť oproti BCH kódu má lepší informační poměr. Např. pro danou délku slova paměti n = 16 je z tohoto počtu 11 informačních bitů při použití rozšířeného Hammingova kódu, zatímco pouze 7 informačních bitů při použití BCH kódu. Naopak pro zakódování zprávy o k = 21 informačních bitech je zapotřebí při použití rozšířeného Hammingova kódu slovo délky 27 bitů, zatímco při použití BCH kódu je zapotřebí slovo délky 31 bitů.

Výhodou BCH kódu pro dvojnásobné opravy je možnost detekování a opravy většího počtu chyb, možnost detekování a opravy shlukových chyb. Dále má větší volitelnost parametru délky slova n, ale tato délka již nemůže být zkrácena o nadbytečné informační bity, neboť výsledný kód by pak již nebyl cyklický.

Vzhledem k tomu, že žádná z předností BCH kódu pro dvojnásobné opravy není požadována, ani nemá zásadní význam pro výběr kódu, jako vhodný kód byl zvolen rozšířený Hammingův kód. Pouze pokud by byl stanoven nezbytný požadavek, aby kód uměl detekovat a opravovat i shlukové chyby, pak by naopak výběrem byl jednoznačně určen BCH kód pro dvojnásobné opravy.

PRAKTICKÁ ČÁST

6 ROZBOR SOUČASNÉHO STAVU

6.1 Kodéry a dekodéry Hammingova kódu

Hammingův kód byl objeven Richardem Hammingem a popsán v roce 1950 v knize *Error Detecting and Error Correcting Codes* [5]. Díky svým vlastnostem je však stále využíván v mnoha aplikacích dodnes. Teorie zabývající se způsobem tvorby generující či kontrolní matice, výpočtem kontrolních bitů i dekódováním slov byla popsána v mnoha publikacích a je možné dohledat velké množství veřejně přístupného software Hammingových kodérů a dekodérů. Stejně tak byl řešen i problém vytvoření aplikace schopné generovat kodér a dekodér Hammingova kódu v jazyku VHDL. Cílem této práce je však vypracovat takové řešení, které bude v souladu s požadavky a vnitřními předpisy zadavatelské firmy ON Design Czech, s.r.o., Brno. Jako příklad jsou zde proto uvedeny dva projekty, které posloužily pro inspiraci k řešení daného problému. Vlastní řešení pak využívá přednosti obou představených projektů.

6.2 Configurable Hamming Generator

Jako první bude uveden projekt pod názvem Configurable Hamming Generator, dostupný na internetových stránkách http://opencores.org/project,hamming_gen, jehož tvůrcem je Alexandre Amory. Tento projekt je svobodným softwarem a je možné jej upravovat a redistribuovat pod podmínkami licence GNU General Public Licence.

V tomto projektu byla vytvořena aplikace v jazyku C++, která generuje jednoduchý knihovní balík (dále také označován jako *package*) v jazyku VHDL, obsahující funkce vykonávající úlohu kodéru a dekodéru Hammingova kódu. Tato aplikace je schopna generovat kodér a dekodér jak pro Hammingův kód pro opravu jedné chyby (dále také označován jako *Single Error Correction; SEC*), tak právě pro rozšířený Hammingův kód pro opravu jednonásobných a detekci dvojnásobných chyb (dále také označován jako *Single Error Correction, Double Error Detection; SEC-DED*). Kódování informačních bitů je prováděno na základě výpočtu pozice jedničkových bitů v kontrolní matici, uspořádané podle tvaru pro systematický kód:

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & | & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & | & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & | & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & | & 1 & 1 & 1 & 1 \end{bmatrix}$$

kdy sloupce této matice tvoří vždy čísla v binárním tvaru, jež jsou seřazeny zprava podle velikosti, přičemž sloupce, jejichž hodnota je mocninou čísla 2 (obsahují tedy pouze jeden jedničkový bit) jsou řazeny na konec matice. Tato matice je rozšířena přidáním nuly na konci každého řádku a řádku samých jedniček. Vzhledem k pravidelnosti uspořádání této matice bylo možné v aplikaci v jazyku C++ vypočíst konkrétní pozice jedničkových bitů a vygenerovat VHDL kód obsahující pouze jejich čísla. Při tomto uspořádání však není řešen problém optimalizace matice, který bude vysvětlen níže.

Při dekódování dat je syndrom získán tak, že se nejdříve z přečtených informačních bitů znovu vypočítají kontrolní bity a ty se pak sečtou modulo 2 s přečtenými informačními bity. Toto řešení je používáno v paměťových systémech, aby se docílilo úspory materiálu, neboť obvody, které realizují rovnice pro výpočet kontrolních bitů, jsou společné pro kodér i dekodér [1]. Avšak při tomto řešení dochází k většímu zpoždění signálu, neboť kvůli opakovanému kódování přečtených dat prochází signál delší kombinační cestou. Pro samotné dekódování syndromu byl v tomto projektu použit algoritmus prohazování sloupců kontrolní matice tak, aby při dekódování odpovídala binární hodnota sloupce jeho pořadí. Ani toto však není vhodné řešení pro implementaci, neboť je náročné na velikost plochy na čipu a oproti navrženému řešení má rovněž delší jeho nejdelší kombinační cestu. Na tento parametr pravděpodobně nebylo pohlíženo z toho důvodu, že kodér i dekodér je vytvořen jako model a není určen pro implementaci. Bylo proto nutné najít jiné řešení.

6.3 Projekt „UMEL_ŠB“

Vedoucí této práce, Ing. Martin Šťáva, Ph.D., řešil tento problém dříve pro projekt „UMEL_ŠB“ a toto řešení poskytl k nahlédnutí při tvorbě této práce. V tomto projektu byl kodér i dekodér popsán parametricky v jazyku VHDL. Způsob výpočtu kontrolních bitů je stejný, jako v předchozím případě, pouze zde probíhá na základě přístupu přímo ke kontrolní matici, jež je uložena v jiném knihovním balíku.

Oproti prvnímu projektu je zde navíc řešen problém optimalizace kontrolní matice. Pro konstrukci generátoru kontrolních bitů či syndromu je výhodné v kontrolní matici minimalizovat počet jedniček, což vede ke zkrácení rovnic pro výpočet jednotlivých kontrolních bitů a tím i zmenšení počtu použitých hradel pro získání těchto bitů. Proto se kontrolní matice konstruuje tak, aby její sloupce měly lichou paritu a minimální počet jedniček a její řádky měly pokud možno stejnou Hammingovu váhu [1]. V případě generace kódu o délce kódových slov rovných mocnině čísla dva, jsou v matici obsaženy všechny nenulové vektory o délce počtu kontrolních bitů a všechny řádky kontrolní matice obsahují stejný počet jedniček. Pokud je však použito méně informačních bitů, než je možné využít pro daný počet kontrolních bitů, projeví se to v kontrolní matici vynecháním některých sloupců. Podle toho, které sloupce kontrolní matice jsou vynechány, budou jednotlivé řádky obsahovat různý počet jedniček. V tomto projektu byla snaha vyvažovat počty jedniček na jednotlivých řádcích kontrolní matice.

7 KODÉR A DEKODÉR HAMMINGOVA KÓDU

7.1 Vlastní řešení problému

Cílem této práce bylo vytvořit aplikaci schopnou generovat kodér a dekodér popsané syntetizovatelným kódem v jazyku VHDL. Jako vhodné řešení se ukázalo zvolit pro realizaci kodéru a dekodéru parametrický popis v jazyku VHDL. Popis vychází z přístupu ke konstantě kontrolní matice kódu, která je uložena spolu s konstantami parametrů v přiloženém knihovním balíku. Tento způsob je vhodný pro implementaci – je zcela syntetizovatelný, univerzální, přehledný, úsporný na použitou plochu na čipu a výsledné řešení má oproti ostatním návrhům kratší maximální kombinační cestu.

Zdrojový kód kodéru, dekodéru a knihovního balíku byl převzat z řešení Martina Šťávy, jež bylo použito v projektu „UMEL_ŠB“. Obsah zdrojových kódů byl následně upraven tak, aby odpovídal pravidlům pro psaní zdrojových kódů, která byla požadována zadavatelskou firmou ON Design Czech. V knihovním balíku byla také upravena kontrolní matice z důvodu její lepší optimalizace a odpovídajícím způsobem byl upraven i kodér a dekodér Hammingova kódu.

Vedle úpravy převzatých modulů byla navíc vytvořena aplikace v jazyku C, která tyto moduly generuje společně s knihovním balíkem konstant parametrů kódu. Součástí aplikace je i řešení optimalizace kontrolní matice.

Pro ověření funkčnosti kodéru a dekodéru bylo vytvořeno verifikační prostředí, jehož součástí je i model programovatelné paměti typu ROM s možností vkládání chyb. V tomto prostředí byla provedena verifikace generovaných syntetizovatelných modulů kodéru a dekodéru. Struktura tohoto prostředí byla vytvořena na základě požadavků zadavatelské firmy ON Design Czech, s.r.o., Brno.

7.2 Syntetizovatelné moduly v jazyku VHDL

Tato část práce se věnuje návrhu syntetizovatelných modulů kodéru a dekodéru popsaných parametricky jazyku VHDL. Parametry tvoří tři konstanty uložené v knihovním balíku a to šířka informačního vektoru *k* (ve zdrojovém kódu je pro lepší přehlednost uváděna jako konstanta *c_INF_DATA_WIDTH*), šířka kontrolního vektoru *r* (ve zdrojovém kódu uváděna jako *c_CHECK_DATA_WIDTH*) a šířka kódového vektoru *n* (ve zdrojovém kódu uváděna jako *c_ENCODED_DATA_WIDTH*).

7.2.1 Kodér Hammingova kódu

Návrh kodéru Hammingova kódu je popsán v souboru *coder.rtl.vhd* a jeho vstupní a výstupní signály (dále textu označovány jako porty) jsou uvedeny v tabulce 4:

Tab. 4: Popis portů dekodéru

Název portu	Typ portu	Počet bitů	Popis portu
DATA_IN	in	*k*	Vstupní informační vektor
DATA_OUT	out	*n*	Výstupní kódový vektor

Jak bylo ukázáno v kapitole 3.4, kódování kontrolních bitů kódového slova může vycházet buď z generující, nebo z kontrolní matice. Obě tato řešení jsou ekvivalentní a jejich výsledkem jsou tytéž rovnice pro výpočet kontrolních bitů. Pro realizaci kodéru bylo zvolen způsob kódování vycházející z kontrolní matice, v simulační části byl pak použit i způsob vycházející z generující matice pro vytvoření modelu kodéru ověřujícího jeho činnost.

Kódování pomocí kontrolní matice vychází z faktu, že pro libovolný kódový vektor v musí platit, že jeho součin s kontrolní maticí je roven nulovému vektoru. Jako konkrétní příklad realizace kodéru bude uvedeno kódování informačního vektoru o délce 4 s optimalizovanou kontrolní maticí $\boldsymbol{H}$, sestavenou pro systematický kód. Při popisu kontrolní matice systematického kódu budou označovány sloupce, jejichž index odpovídá informačním bitům kódového vektoru jako informační sloupce matice. Podobně sloupce, jejichž index

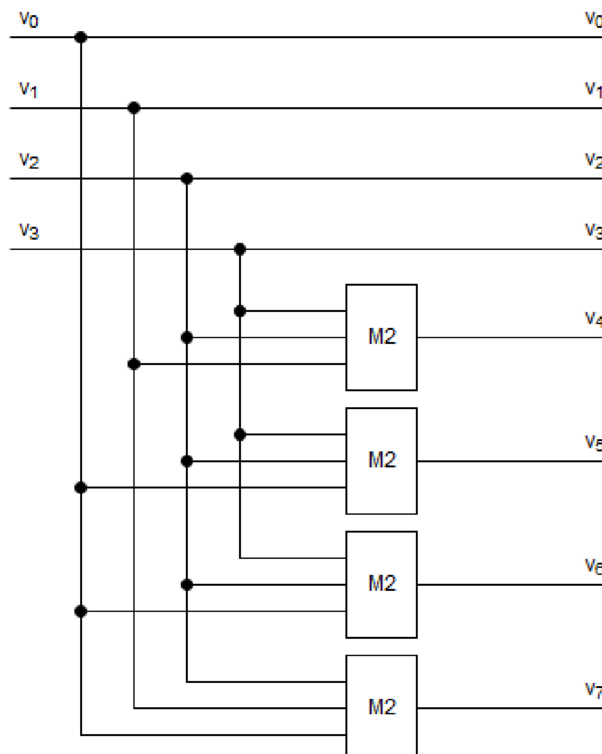
odpovídá kontrolním bitům kódového vektoru, budou označovány jako kontrolní sloupce matice.

$$\boldsymbol{H} = \begin{bmatrix} 0 & 1 & 1 & 1 & | & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & | & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & | & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & | & 0 & 0 & 0 & 1 \end{bmatrix}$$

Při rozepsání jednotlivých skalárních součinů obecného vektoru a kontrolní matice dospějeme k rovnicím pro jednotlivé kontrolní bity. Při vstupním vektoru $\boldsymbol{v} = (v_0, v_1, v_2, v_3)$ tak získáme pro kontrolní bity v_4, v_5, v_6, v_7 rovnice:

$$\begin{aligned} v_1 + v_2 + v_3 &= v_4 \\ v_0 + v_2 + v_3 &= v_5 \\ v_0 + v_1 + v_3 &= v_6 \\ v_0 + v_1 + v_2 &= v_7 \end{aligned}$$

Kodér Hammingova kódu tedy sestává pouze z generátoru sudých parit sestavených podle uvedených rovnic, tedy jako součet modulo 2 (jež mohou být realizována hradlem *xor*) těch bitů informačního vektoru, jejichž pozice odpovídá pozici nenulových bitů na tom řádku kontrolní matice, který odpovídá pozici kontrolního bitu. V případě uvedené kontrolní matice tak matice vnikne zapojení, uvedené na obrázku 5:



Obr. 5: Kodér rozšířeného Hammingova kódu (8,4)

Z těchto poznatků vychází parametrický popis navrženého kodéru rozšířeného Hammingova kódu. Ve zdrojovém kódu je za návěstím *parity_gen* příkazem *for-generate* generován pro každý kontrolní bit (*parity_bit*) obvod, realizující rovnici pro výpočet hodnoty daného kontrolního bitu. Toho je docíleno procházením řádků kontrolní matice uložené v

konstantě *c_H_EXT*. Při detekci jedničkového bitu v kontrolní matici je proveden součet modulo 2 mezivýsledku uloženého do proměnné p a toho bitu informačního vektoru, jehož pozice odpovídá pozici nalezeného jedničkového bitu v kontrolní matici. Syntetizátor pak pouze seskládá jednotlivé kroky výpočtu do výsledného obvodu obsahujícího sčítačky modulo 2 správných bitů kontrolní matice. Kontrolní bity jsou poté přiřazeny za vstupní informační vektor, takže kódový vektor má na horních k bitech informační bity a na dolních r bitech kontrolní bity.

7.2.2 Dekodér Hammingova kódu

Dekodér Hammingova kódu je popsán v souboru *decoder.rtl.vhd* a jeho porty jsou uvedeny v tabulce 5

Tab. 5: Popis portů dekodéru

Název portu	Typ portu	Počet bitů	Popis portu
DATA_IN	in	n	Vstupní kódový vektor
DATA_OUT	out	k	Výstupní dekódovaný informační vektor
ERR_DET	out	1	Příznak o detekování dvojnásobné chyby v kódovém vektoru
ERR_COR	out	1	Příznak o opravení jednonásobné chyby v kódovém vektoru

Dekódování slov je opět založeno na vlastnostech kontrolní matice vysvětlených v kapitole 3.4. Nejdříve je součinem kódového slova s kontrolní maticí vypočten syndrom. Podobně jako u kódování kontrolních bitů získáme rozepsáním jednotlivých skalárních součinů rovnice pro výpočet jednotlivých bitů syndromu. Pro optimalizovanou kontrolní matici uvedenou výše bychom tak získali následující rovnice:

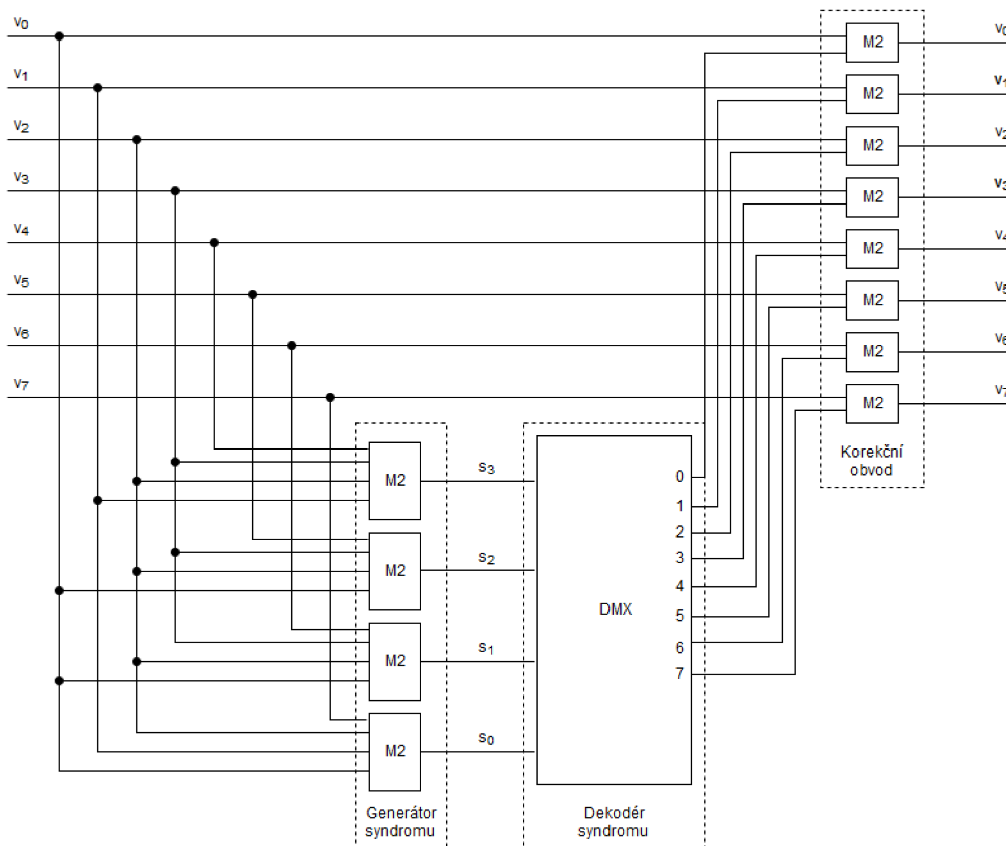
$$s_3 = v_1 + v_2 + v_3 + v_4$$

$$s_2 = v_0 + v_2 + v_3 + v_5$$

$$s_1 = v_0 + v_1 + v_3 + v_6$$

$$s_0 = v_0 + v_1 + v_2 + v_7$$

Dekodér obecně sestává z obvodu pro výpočet syndromu, převodníku syndromu na chybový vektor a z korekčního obvodu [1, s. 134].



Obr. 6: Dekodér rozšířeného Hammingova kódu (8,4)

Ve zdrojovém kódu je v části za návěstím *syndrome_gen* pomocí příkazu *for-generate* pro každý bit syndromu vytvořen obvod, sestavený z hradel *xor*, který realizuje rovnici pro výpočet bitu syndromu. Toho je opět docíleno procházením řádků kontrolní matice uložené v konstantě *c_H_EXT*. Při detekci jedničkového bitu v kontrolní matici je proveden součet modulo 2 mezivýsledku uloženého do proměnné *p* a toho bitu kódového vektoru, jehož pozice odpovídá pozici nalezeného jedničkového bitu v kontrolní matici. Syntetizátor pak pouze seskládá jednotlivé kroky výpočtu do výsledného obvodu obsahující sčítačky modulo 2 správných bitů kontrolní matice. Syndromy jednonásobných chyb přitom mají v kódu definovaném optimalizovanou kontrolní maticí lichý počet jedničkových bitů a jejich hodnota odpovídá některému ze sloupců kontrolní matice, zatímco syndromy dvojnásobných chyb mají sudý počet jedniček a jejich hodnota neodpovídá žádnému ze sloupců kontrolní matice. To je dáno tím, syndrom dvojnásobné chyby je součtem modulo 2 po souřadnicích syndromů jednonásobných chyb [1]. Dekodér obsahuje tedy jednoduchý obvod, popsaný procesem *p_corr_err*, který provedením součtu modulo 2 všech bitů syndromu určí jeho paritu, a tak zjistí, zda případná chyba byla jednonásobná, či dvojnásobná.

Dále je provedeno dekódování syndromu, které je založeno na porovnávání jeho hodnoty s jednotlivými sloupci kontrolní matice. Při výskytu jedné chyby v kódovém vektoru odpovídá pozice chybového bitu pozici toho sloupce, jehož hodnota se rovná hodnotě syndromu. Syndrom je však porovnán pouze s těmi sloupci kontrolní matice, jejichž pozice odpovídá pozici informačních bitů v kódovém vektoru. Pokud totiž nastala jednonásobná chyba v jednom z kontrolních bitů kódového vektoru, tato chyba je odhalena na základě rozpoznání nenulového syndromu s lichým počtem jedniček. Přesnou pozici chybného bitu však není třeba znát, neboť na výstup se přenesou pouze informační bity. V případě jednonásobné chyby v jednom z informačních bitů je porovnáním syndromu získám korekční vektor jeho součtem modulo 2 s přijatým vektorem je získán opravený informační vektor.

7.2.3 Knihovní balíky *hamm_rom_xb_pkg*

Aplikace *Hamming_gen* v jazyku C generuje dva knihovní balíky, z nichž jeden slouží k předání důležitých konstant parametrů pro syntetizovatelné moduly kodéru a dekodéru a druhý pro předání těchto parametrů pro simulační funkční modely (dále jen *Bus functional model; BFM*) verifikačního prostředí. Název knihovních balíků obsahuje číslo, udávající šířku kódového vektoru, kterým je nahrazeno písmeno „*x*“. Název knihovního balíku, který je určený pro funkční modely má oproti knihovnímu balíku, určenému pro syntetizovatelné moduly navíc předponu *bfm_**. Obsah obou souborů je však až na název samotného knihovního balíku zcela identický. Důvodem generování dvou souborů je požadavek firmy ON Design Czech, neboť je vhodné, aby soubory, určené pro simulaci, mohly být zcela odděleny od souborů, určených pro syntézu.

Všechny konstanty jsou pojmenovány ve shodě s požadavky firmy ON Design Czech, tedy mají prefix *c_** a jejich název je psán velkými písmeny. Parametry kódu jsou šířka informačního vektoru *k*, pojmenovaná *c_INF_DATA_WIDTH*, šířka kontrolního vektoru *r*, pojmenovaná *c_CHECK_DATA_WIDTH* a šířka kódového vektoru *n*, pojmenovaná *c_ENCODED_DATA_WIDTH*, která je daná součtem předchozích dvou parametrů. Tyto konstanty jsou vypočteny a generovány aplikací v jazyku C. Kontrolní matice je rovněž zadána jako konstanta a je pojmenována *c_H_EXT*.

.

7.3 Aplikace generující syntetizovatelné moduly

V této části práce je popsána aplikace v jazyku C, která generuje syntetizovatelné moduly kodéru, dekodéru a knihovní balíky určené po syntézu i simulaci. Dále je zde vysvětlen způsob optimalizace kontrolní matice a popsán algoritmus, který generuje optimalizovanou kontrolní matici.

Funkcí této aplikace je na základě vstupní informace o šířce informačního vektoru vypočíst parametr šířky kontrolního vektoru, vygenerovat optimalizovanou kontrolní matici Hammingova kódu a vytvořit dva knihovní balíky popsané v jazyku VHDL, které budou obsahovat konstanty parametru šířky informačního i kontrolního vektoru a vygenerovanou kontrolní matici rozšířeného Hammingova kódu. Pro tuto aplikaci byl zvolen jazyk C, neboť operace jsou vcelku jednoduché a nevyžadují speciální funkce jiných programovacích jazyků.

Aplikace má název *hamming_gen* a nemá grafické rozhraní, je spustitelná pouze z příkazové řádky operačního systému. Při spuštění přijímá hlavní funkce *main* dva vstupní parametry a kontroluje, zda nebyly zadány více než dva, či méně než jeden parametr. U zadaných parametrů záleží na pořadí jejich zadání. První přijímaný parametr je povinný a je jím požadovaná šířka informačního vektoru. Je kontrolováno, zda je tímto parametrem řetězec obsahující čistě číselnou hodnotu a navíc, zda tato hodnota není menší než číslo dva. Rozšířený Hammingův kód, který má pouze jeden informační bit, je vlastně opakovacím kódem délky čtyři a proto není předpokládáno použití kodéru a dekodéru rozšířeného Hammingova kódu tento jednoduchý kód. Druhým nepovinným parametrem aplikace je předpona výstupního souboru. Název výstupních souborů je předdefinován a je jím řetězec *hamm_rom_xb_pkg*, v případě knihovního balíku určeného pro kodér a dekodér a *bfm_hamm_rom_xb_pkg* v případě knihovního určeného pro funkční modely verifikačního prostředí, kde písmeno „*x*“ je nahrazeno délkou kódového vektoru daného kódu.

V programu jsou definovány tři pomocné funkce vycházející z cílů této aplikace a to funkce *get_parity_bits* pro výpočet počtu potřebných kontrolních bitů při daném počtu informačních bitů, funkce *generate_opt_matrix* pro generaci optimalizované kontrolní matice do jednorozměrného pole a funkce *write_vhdl* pro vytvoření výstupních souborů obsahující knihovní balík s parametry kódu a kontrolní maticí.

Funkce *get_parity_bits* pro výpočet kontrolních bitů vychází z obecného vzorce pro délku rozšířeného Hammingova kódu, uvedených v kapitole 3.4.:

$$n = k + r \tag{7.1}$$

$$n \leq 2^{r-1} \tag{7.2}$$

kde n značí délku kódového vektoru, r počet kontrolních bitů a k počet informačních bitů. Z těchto rovnic není možné vyjádřit minimální hodnotu počtu kontrolních bitů r, pouze v závislosti na parametru počtu informačních bitů k. Funkce je proto založena na cyklickém zvyšování čísla r a vypočtení skutečné délky kódového vektoru n pomocí rovnice (7.1) do ověření platnosti rovnice (7.2).

Stěžejní částí je generace optimalizované kontrolní matice rozšířeného Hammingova kódu, zajištěné funkcí *generate_opt_matrix*. Jak bylo řečeno dříve, pro konstrukci generátoru kontrolních bitů či syndromu je výhodné v kontrolní matici minimalizovat počet jedniček, neboť to vede ke zmenšení počtu použitých hradel potřebných pro získání kontrolních bitů při kódování kontrolních bitů či bitů syndromu. Optimalizovaná kontrolní matice se konstruuje tak, aby její sloupce měly lichou paritu a minimální počet jedniček a její řádky měly pokud možno stejnou Hammingovu váhu [1]. V případě generace rozšířeného Hammingova kódu o délce kódových slov rovných mocnině čísla dva, jsou v matici obsaženy všechny nenulové vektory o délce počtu kontrolních bitů, jež jsou rozšířeny o jedničkový bit a všechny řádky kontrolní matice až na poslední obsahují stejný počet jedniček. Provedením elementární úpravy součtu všech řádků kontrolní matice s posledním řádkem docílíme liché parity všech sloupců a poslední řádek bude mít stejný počet jedničkových bitů, jako ostatní řádky. Pokud je však použito méně informačních bitů, než je možné využít pro daný počet kontrolních bitů, projeví se to vynecháním některých sloupců kontrolní matice. Podle toho, které sloupce kontrolní matice nejsou využity pro její vytvoření, budou jednotlivé řádky obsahovat různý počet jedniček.

Výslednou kontrolní matici tedy budou tvořit vždy sloupce s lichým počtem jedniček, přičemž sloupce obsahující pouze jednu jedničku budou stavěny na konec matice, pro sestavení systematického kódu. Z množiny všech nenulových sloupců, s lichým počtem jedniček, které by tvořili kontrolní matici kódu o délce $n = 2^{r-1}$ jsou postupně vybírány sloupce tak, aby vždy součet jedniček na jednotlivých řádcích nelišil více než o jednu jedničku.

Algoritmus se skládá z 8 základních kroků:

1) Vytvoření pole *max_k_matrix* obsahující hodnoty informačních sloupců kontrolní matice Hammingova kódu o délce $n = 2^{r-1}$.
2) Úprava všech hodnot tohoto pole tak, že je binární tvar tohoto čísla posunut o jeden bit vlevo (vynásoben dvěma) a poslední bit doplněn tak, aby binární číslo mělo lichou paritu.
3) Vytvoření pole *max_k_matrix_num_ones* s informací o počtu jedniček každého čísla pole *max_k_matrix* a pole *max_k_matrix_col_used*, obsahující informaci, zda byl daný sloupec použit pro generaci výstupní kontrolní matice.

4) Vytvoření pole *parity_col* obsahujícího informaci o tom, zda je v některém z řádků výsledné kontrolní matice více jedniček než v některém z jiných řádků.
5) Prohledávání pole *max_k_matrix* a hledání hodnoty takového sloupce, který nebyl dosud použit. Z těchto sloupců je vybírán sloupec, jehož hodnota v binárním tvaru má jedničky na těch pozicích, které odpovídají řádkům výsledné kontrolní matice s menším počtem jedniček než je na ostatních řádcích.
6) Uložení nalezeného sloupce výsledné matice *matrix* a zároveň uložení informace o tom, že byl daný sloupec již použit do pole *max_k_matrix_col_used*.
7) Opakování 5. a 6. kroku algoritmu do naplnění všech informačních sloupců výsledné matice.
8) Doplnění kontrolních sloupců výsledné matice. Tuto část tvoři binární hodnoty čísel 2^n pro všechna přirozená čísla $n < r$. V této části je proto matice tvořena diagonálou jedniček, což odpovídá tvaru kontrolní matice systematického kódu.

.

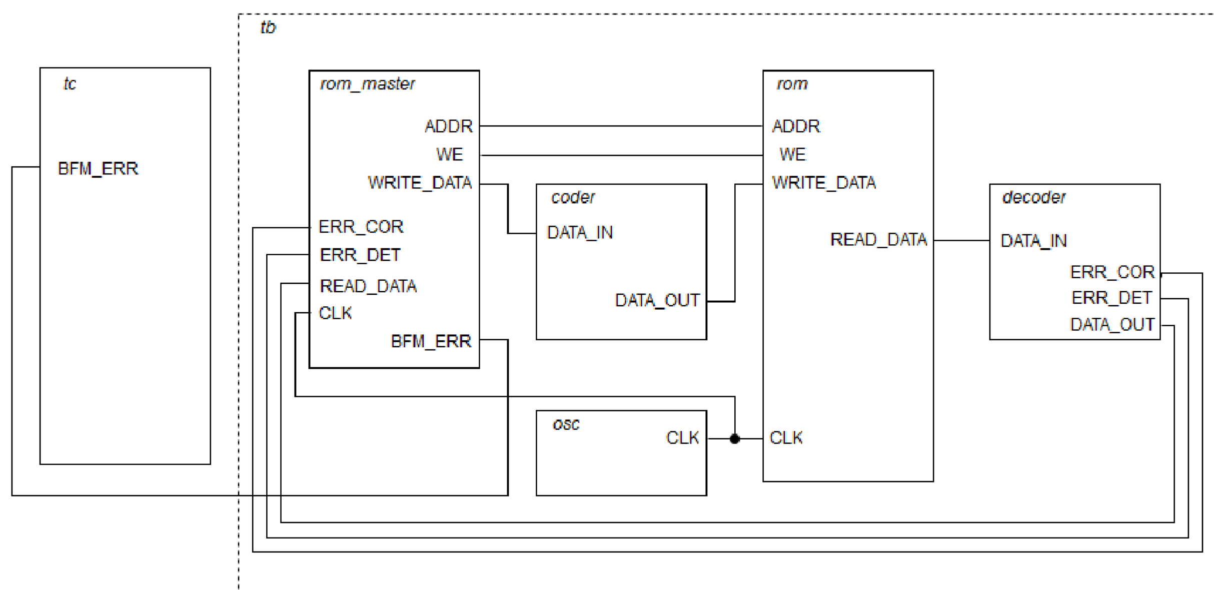
Výstupem funkce *generate_opt_matrix* je jednorozměrné pole, obsahující vygenerovanou kontrolní matici rozšířeného Hammingova kódu

Funkce *write_vhdl_pkg* pouze vytvoří textový soubor typu vhd a zapíše do něj statickou část VHDL kódu s deklarací knihovního balíku, konstant parametrů kódu a konstanty kontrolní matice, uložené do dvourozměrné struktury. Těmto konstantám dynamicky přiřadí příslušné hodnoty, které jsou parametry této funkce.

8 VERIFIKAČNÍ PROSTŘEDÍ

8.1 Struktura prostředí

Proces verifikace slouží k ověření, zda obvod splňuje zadaná kritéria. Existuje více metodologií pro verifikaci, v tomto případě byla validace provedena podle zásad zadavatelské firmy ON Design Czech. Schéma verifikačního prostředí je uvedeno na obrázku 7.



Obr. 7: Struktura verifikačního prostředí

Modul *testbech* (označený ve zdrojových kódech *tb*) tvoří vrcholnou jednotku verifikačního prostředí. To je ovládáno jednotlivými architekturami modulu *testcase* (označené ve zdrojovém kódu *tc*), které mohou být připojeny k verifikačnímu prostředí. Každý architektura modulu *testcase* slouží pro ověření funkčnosti jednoho objektu DUV (*Design Under Verification*), jimiž jsou v tomto návrhu kodér a dekodér.

V modulu *testcase* jsou volány pouze procedury funkčního modelu (dále také *BFM – Bus functional model*) řídící jednotky paměti *rom_master,* uložené v knihovním balíku *rom_master_pkg*, které ovládají celý zbytek prostředí. Pojmem BFM myslíme modely určené pouze pro simulaci, které modelují vstupně-výstupní chování určité části návrhu, bez popisu jeho syntetizovatelné struktury. Mohou být ovládány jak přes vstupní a výstupní signály (*porty*), tak pomocí procedur. Navržené verifikační prostředí obsahuje dva hlavní BFM a to model programovatelné paměti typu ROM, pojmenovaný *rom,* a model řídící jednotky paměti pojmenovaný *rom_master*. Ke každému z těchto modelů je přiřazen knihovní balík (*package*) obsahující procedury, kterými je model ovládán. Procedury, které jsou uloženy v knihovním balíku *rom_master_pkg* tak můžou řídit paměť dvěma způsoby – buď synchronním způsobem, kdy stimulují signály na výstupu řídící jednotky *rom_master*, nebo asynchronním způsobem, kdy přistupují pomocí procedur uložených v knihovním balíku *rom_pkg* přímo k paměti. Stejně tak komunikaci v celém systému můžeme rozdělit do dvou kategorií – komunikace probíhající skrz rozhraní všech modulů, či komunikace probíhající mimo něj.

Verifikační prostředí je navrženo tak, aby simulovalo činnost celého systému. Komunikace skrz rozhraní tak může probíhat následujícím způsobem. Model *rom_master* slouží jako řídící jednotka paměti a signálem *WE* dává pokyn k zápisu dat do paměti, nastavuje signál *WRITE_DATA* s daty, jež mají být do paměti zapsány a signál *ADDR* s hodnotou adresy, na kterou mají být data uložena. Data, jež mají být zapsána, jsou kódována v kodéru a celý kódový vektor je v případě aktivního povolovacího signálu *WE* synchronně s nástupnou hranou hodinového signálu uložen do paměti. Čtení paměti probíhá na rozdíl od zápisu s každou nástupnou hranou hodinového signálu. Adresa paměti, která má být přečtena je opět nastavena řídící jednotkou paměti *rom_master.* Tento model je navíc schopen vložit do paměti určitý počet chyb na přesně daná místa kódového vektoru. Přečtená data jsou dekódována pomocí dekodéru a výstupní informační vektor je přijat zpět do řídící jednotky *rom_master* zároveň se signály příznaků o opravení jednonásobné chyby, či detekci dvojnásobné chyby. Porovnáním dat odeslaných pro zápis a dat přijatých je možné v tomto modelu ověřit správnou součinnost obou DUV.

V případě testování celého systému najednou by však mohla nastat situace, kdy nesprávnou činností dekodéru nikdy nemusí být odhalena chyba kodéru a naopak nesprávnou činností kodéru nikdy nemusí být odhalena chyba dekodéru. Jedním ze základních požadavků je proto možnost ověření funkčnosti jednoho DUV bez nutnosti využití jiného. Byly tedy řešeny následující problémy:

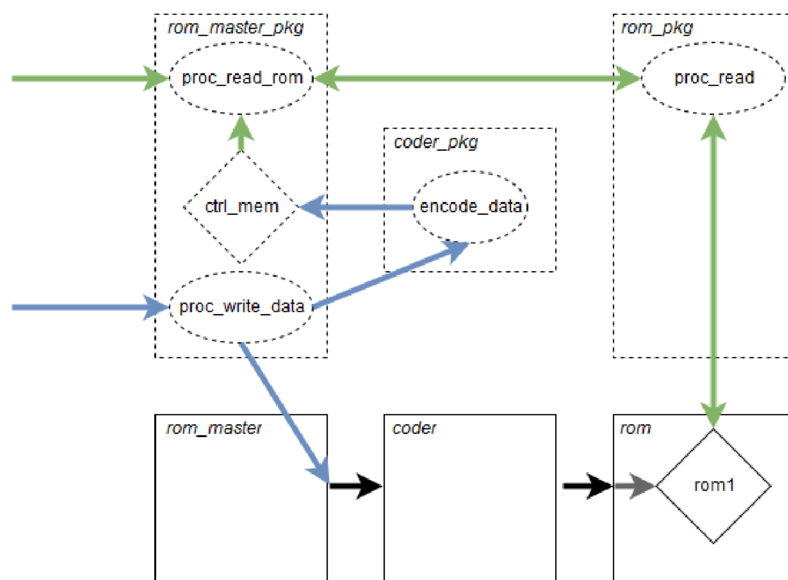
1) Pro ověření funkčnosti kodéru je nutné bez použití modulu dekodéru zkontrolovat, zda kontrolní bity uložené do paměti jsou vypočítány správně.

2) Pro ověření funkčnosti dekodéru je nutné bez použití modulu kodéru zkontrolovat, zda data vyčtená z paměti jsou správně dekódována a je opraven či detekován správný počet chyb.

Z výše uvedených podmínek vyplývá, že kromě komunikace přes rozhraní je nutné použít komunikaci mezi moduly mimo rozhraní. Každý BFM by přitom měl mít možnost

komunikovat mimo rozhraní pouze s procedurami deklarovanými v jednom knihovním balíku, jež bude k tomuto BFM připojen. Tento postup je založen na dvou dalších požadavcích. Prvním je, aby celá činnost systému mohla být řízena pouze voláním procedur, uložených v knihovním balíku *rom_master_pkg*. Druhým požadavkem je pak, aby paměť *rom1* byla přístupná pouze v BFM *rom* a procedury v jeho knihovním balíku vykonávaly pouze základní operace zápisu do paměti a čtení z paměti. Pro testování obou modulů byly zvoleny postupy popsané níže.

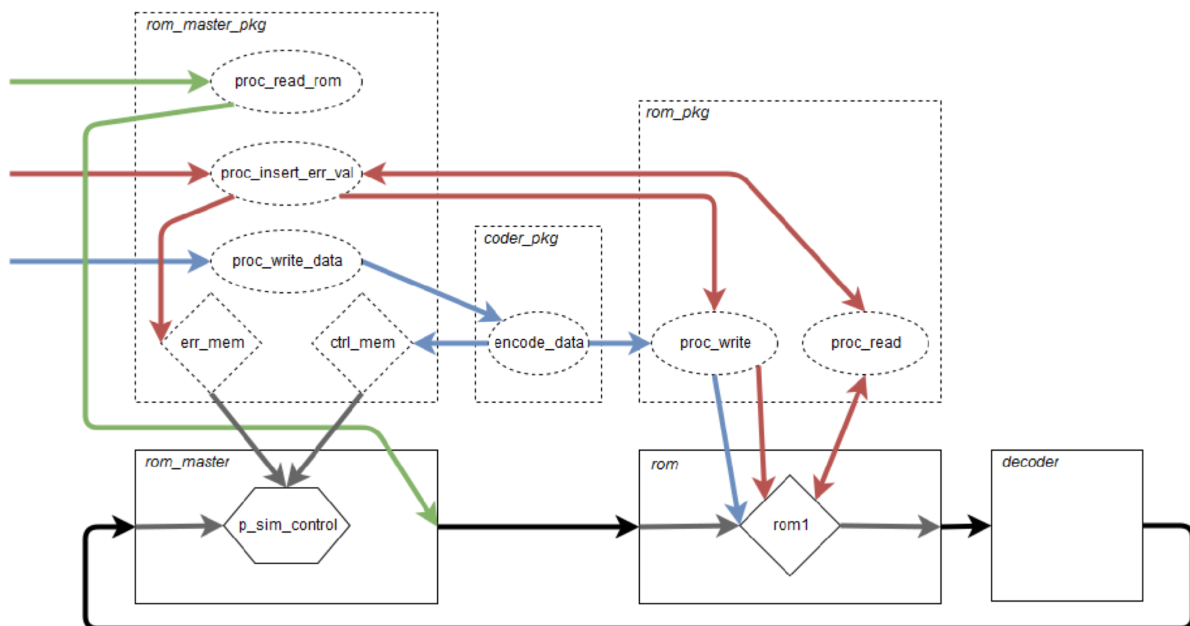
Pro ověření zápisu dat do paměti pomocí kodéru byla zvolena metoda, kdy při kódování informačních bitů jsou tato data kódována paralelně jiným způsobem, a vzájemným porovnáním výsledků je ověřena správnost výsledného kódového vektoru. Celý proces testování zápisu do paměti pomocí kodéru je graficky znázorněn na obrázku 8 a skládá se ze dvou kroků. Prvním krokem (označeným modrými šipkami) je zápis dat do paměti, uskutečněný procedurou *proc_write_data*, která je volána z modulu *testcase*. Tato procedura stimuluje výstupní signály BFM *rom*_master a nastaví na ně požadované hodnoty pro zápis. Data jsou přijata kodérem, který provede výpočet kontrolních bitů a celý kódový vektor je následně uložen do paměti *rom1* v BFM *rom*. Zároveň jsou data kódována pomocí funkce *encode_data* a originální informační vektor je společně s vypočtenými bity uložen do kontrolní paměti *ctrl_mem* v knihovním souboru *rom_master_pkg*. Druhým krokem (označeným zelenými šipkami) je vyčtení dat z paměti *rom1* a jejich porovnání s daty v kontrolní paměti. Vyčtení nemůže být provedeno skrz rozhraní, neboť by pak musel být použit modul dekodéru. Pro tento účel proto slouží procedura *proc_read_rom,* volaná opět z modulu *testcase*. Ta pomocí procedury *proc_read* uložené v knihovním balíku *rom_pkg* vyčte hodnotu z požadované adresy paměti a porovná ji s hodnotou uloženou v kontrolní paměti. Pokud se tyto hodnoty nerovnají, je ohlášena chyba.



Obr. 8: Výměna dat při ověřování funkčního zápisu do paměti pomocí kodéru

Testování správného dekódování dat se skládá ze tří kroků a je celý opět graficky znázorněn na obrázku 9. Prvním krokem (označeným opět modrými šipkami) je zápis dat do paměti *rom1*. Ten nemůže proběhnout skrz rozhraní, neboť pak by musel být použit modul kodéru. Pro tento účel proto slouží procedura *proc_write_data*. Z informačních bitů jsou vypočteny kontrolní bity za pomoci funkce encode_data a celý kódový vektor je pomocí procedury *proc_write* zapsán do paměti *rom1* a zároveň uložen do kontrolní paměti *ctlr_mem.* Druhým krokem (označeným červenými šipkami) je vložení chybové hodnoty do paměti.

K tomu slouží procedura *proc_inser_err_val.* Ta pomocí procedury *proc_read*, uložené v knihovním balíku *rom_pkg*, vyčte hodnotu z adresy paměti, do níž má být chyba vložena. Tato data pak sečte modulo 2 s chybovým vektorem a výsledek uloží do paměti *rom1* pomocí procedury *proc_write* uložené v knihovním balíku *rom_pkg*. Pro uchování informace, kolik chyb se nachází na každé z adres paměti, je vytvořena chybová paměť *err_mem*. Z této paměti je vyčten počet chyb, jež byly dříve vloženy na tyto adresy paměti, a následně vložen aktuální počet chyb. Chybová paměť rozeznává čtyři stavy – žádnou chybu, jednonásobnou chybu, dvojnásobnou chybu a chybu vyšší násobnosti. Třetím krokem (označeným zelenými šipkami) je vyčtení dat z paměti. Procedura *proc_read_rom* stimuluje výstupní signál *ADDR*, kterým je nastavena adresa paměti *rom1*, jež má být vyčtena skrze rozhraní. Vyčtená data jsou dekódována dekodérem a poslána společně s příznaky o počtu opravených či detekovaných chyb poslána skrze rozhraní do modulu *rom_master.* Zde je kontrolováno, zda tato dekódovaná data odpovídají originálním datům, uloženým v kontrolní paměti, a zda příznaky odpovídají počtu chyb, uložených v chybové paměti.



Obr. 9: Výměna dat při ověřování správného dekódování dat přečtených z paměti pomocí dekodéru

Návrh jednotlivých simulačních modelů vychází z výše uvedených požadavků a způsobů testování a je detailně popsána v následujících podkapitolách.

8.2 Zásady psaní zdrojových kódů

Před popisem jednotlivých modelů budou uvedeny některé zásady psaní zdrojových kódů, které byly dodržovány v této práci. Většina z nich byla požadavkem zadavatelské firmy ON Design Czech, některé byly zvoleny pro lepší přehlednost kódu.

Všechny názvy identifikátorů jsou psané malými písmeny. Velkými písmeny jsou značeny pouze názvy konstant, generických konstant a portů (vstupů a výstupů) návrhové jednotky entity. Všechny konstanty jsou uváděny s předponou *c_**, generické konstanty pak s předponou *g_**. Všechny procesy mají předponu *p_** a všechny procedury mají předponu *proc_**. V celém návrhu je udržováno konzistentní pořadí bitů vektorů podle jejich binární váhy, kdy nejvýznamnější bit (dále jen *MSB – Most Significant Bit*) je bit nejvíce vlevo a

nejméně významný bit (*LSB – Least Significant Bit*) je bit nejvíce vpravo. Všechny signály v návrhu jsou tzv. „active high“ a mluvíme-li tedy o jejich aktivní hodnotě, je tím myšlena hodnota ‘1’.

Pro označování souborů a architektur modulů jsou dodržovány tyto zásady. Všechny knihovní balíky (*package*) mají příponu **_pkg*. Všechny soubory určené pro syntézu mají název zakončený příponou **.rtl* a jejich architektury jsou rovněž nazvány *rtl*, pro vyjádření popisu kódu na úrovni meziregistrových přenosů (*RTL – Register Transfer Level*). Všechny modely určené pro simulaci mají název zakončený příponou **.beh* a jejich architektury mají také název *beh*.

Verifikační prostředí je navrženo tak, aby změna každého parametru ovlivňujícího jeho činnost mohla být provedena pouze změnou jedné konstanty na jednom místě. Konstanty určující časové parametry modelů jsou uloženy v knihovním balíku *timing_pkg*. Konstanty určující vlastnosti samotné programovatelné paměti typu ROM jsou uloženy v jejím knihovním balíku *rom_pkg*. Konstanty určující vlastnosti samotného kódu jsou uloženy zvlášť pro soubory určené pro simulaci v knihovním balíku *bfm_hamm_rom_xb_pkg* a zvlášť pro soubory určené pro syntézu v knihovním balíku *hamm_rom_xb_pkg*, kde písmeno *x*, je nahrazeno číslem určující šířku *n* kódového vektoru.

Všechny BFM, jejichž vstupní signály jsou přivedeny z obvodů, určených pro syntézu, obsahují proces *p_x_check*, který kontroluje, zda tyto signály nepřešly do nedefinované hodnoty (hodnota ‘*X*’ výčtového typu *std_logic*). Tato hodnota je zjišťována pomocí funkce *is_x*(<*název_signálu*>), která vrací hodnotu výčtového typu *boolean*. Všem vnitřním signálům BFM jsou přiřazeny implicitní hodnoty, aby se zamezilo nedefinovaným hodnotám na jejich výstupu.

8.3 Model programovatelné paměti typu ROM

Jedním z hlavních cílů této práce bylo navrhnout a v jazyku VHDL popsat model programovatelné paměti typu ROM s možností vkládání chyb. Pro tento model byla zvolena struktura skládající se ze dvou částí – knihovního balíku *rom_pkg* a funkčního modelu *rom.beh*, popisující vstupně-výstupní chování tohoto BFM. Požadovaná funkce toho modelu vychází z řešení problémů verifikace systému popsaných v podkapitole 8.1 a může být shrnuta do těchto bodů:

1) V případě požadavku na zápis provedení synchronního zápisu vstupních dat z kodéru do struktury paměti, ve které budou data uchována pro pozdější vyčtení.

2) Provádět synchronní čtení dat paměti a posílat tato data na výstup modelu s možností nastavení adresy, jež má být přečtena pomocí vstupního signálu

3) Možnost okamžité změny hodnoty uložené v na určité adrese struktury paměti, beze změny hodnoty vstupních signálů

4) Možnost okamžitého vyčtení dat z určité adresy paměti, beze změny vstupních signálů

5) Kontrolovat, zda výstupní signály z kodéru nenabývají nedefinovaných hodnot.

Vstupní a výstupní signály modelu jsou uvedeny v tabulce 6:

Tab. 6: Popis portů modelu *rom*

Název portu	Typ portu	Počet bitů	Popis portu
CLK	in	1	Hlavni hodinový signál
WRITE_DATA	in	*n*	Vstupní kódový vektor, jež má být zapsán do paměti
WE	in	1	Povolovací signál „write enable“ umožňující zápis dat do paměti
ADDR	in	*m*	Číslo adresy paměti, jež má být přečtena či přepsána
READ_DATA	out	*n*	Výstupní kódový vektor s přečtenými daty z paměti

Z důvodů potřeby přistupovat k paměti z různých procesů byl pro paměť zvolen typ objektu sdílená proměnná (*shared variable*), který takový přístup umožňuje. Paměť nese název *rom1* a je deklarována jako sdílená proměnná typu *t_rom_type*, což je vlastní typ obsahující dvourozměrnou strukturu vektorů výčtového typu s*td_logic_vector*. Adresa paměti je do modulu předávána jako vektor o maximální šířce definované konstantou *c_ROM_ADDR_WIDTH* (v tabulce portů pojmenované jako *m*), která je uložena v knihovním balíku paměti *rom_pkg*. Počet adres paměti tedy odpovídá mocnině čísla dva na hodnotu této konstanty. Šířka buňky paměti pak odpovídá šířce kódového vektoru *n*.

Operace čtení z paměti a zápis do paměti vychází z popisu funkce paměti a proto mohou být provedeny dvěma způsoby. První je pomocí procesu *p_rom_ctrl*, který modeluje synchronní chování paměti v závislosti na vstupních signálech BFM. S nástupnou hranou hodinového signálu *CLK* je porovnána hodnota signál *WE* (*write enable*), povolující zápis do paměti. Pokud se tento signál nachází v aktivní hodnotě, jsou vstupní data BFM *WRITE_DATA* zapsána na adresu paměti vybranou vstupním signálem *ADDR*. Následně jsou tatáž data z paměti přečtena a předána na výstupní signál *READ_DATA*. Pokud je signál *WE* neaktivní, je zápis do paměti vynechán a s nástupnou hranou hodinového signálu *CLK* se provede pouze vyčtení hodnoty z adresy definované vstupním signálem *ADDR*.

Vzhledem k synchronnímu zápisu dat do paměti je nutné řešit otázku metastability. Pokud by nová data přišla v době náběžné hrany hodinového signálu, do paměti by se uložily nedefinované hodnoty a obvod by mohl přejít do metastabilního stavu. Aby tato situace nenastala, je potřeba ošetřit, aby vstupní data zůstala stabilní po určitou dobu t_s před příchodem náběžné hrany hodin a po určitou dobu t_h po příchodu náběžné hrany hodin. Časová konstanta t_s se nazývá doba předstihu (*setup time*) a časová konstanta t_h se nazývá doba přesahu (*hold time*). Při dodržení doby předstihu a přesahu je zaručeno, že poslední změna na vstupu bude řádně přenesena na výstup, který pak zůstane konstantní až do dalšího hodinového pulzu [10]. Zaručení této podmínky není řešeno v tomto modelu, ale je ošetřeno v modulu *rom_master*.

Druhým způsobem vykonání operací čtení a zápisu je použití procesu *p_rom_set*. Tento proces slouží k vykonání příslušných operací asynchronně, na základě volání procedur, uložených v knihovním balíku *rom_pkg*. Pro vzájemnou komunikaci mezi knihovním balíkem *rom_pkg* a BFM *rom* je využit princip korespondenčního provozu (též provoz dotaz-odpověď, anglicky *handshake*), který zavádí dva signály pro řízení vzájemné spolupráce (dále jen *handshake* signály) [10]. Proces *p_rom_set* je citlivý na změnu handshake signálu *rom_set_req*, který oznamuje událost vykonávání procedury v knihovním balíku *rom_pkg*. Při změně se proces spustí a vykoná požadovanou operaci zápisu do paměti, či vyčtení hodnoty z paměti. Na konci procesu je znegován druhý handshake signál *rom_set_req_help* a tím je proceduře oznámena událost dokončení vykonávání procesu. Oba handshake signály jsou uloženy právě v knihovním balíku *rom_pkg* Pro předání hodnot požadované adresy a dat pro

zápis, či vyčtených dat je vždy použit princip uložení těchto hodnot do sdílené proměnné *rom_set_var,* deklarované také v knihovním balíku *rom_pkg*. Tento princip však nemůže být použit pro samotné handshake signály, neboť proces v BFM může být citlivý pouze na změnu signálu, nikoli na změnu proměnné. Proměnná *rom_set_var* typu záznam obsahuje prvek *addr*, pro předání hodnoty adresy, do níž má být proveden zápis nebo ze které mají být vyčtena data, dále prvek *data* pro předání dat zapsaných do paměti nebo vyčtených z paměti a prvek *rom_set_op*, který nastavuje v procesu operaci, jež má být vykonána.

V knihovním balíku jsou definované dvě procedury pro okamžité operace s pamětí. Procedura *proc_write* slouží k zápisu požadovaných dat do paměti. Při jejím volání přijímá parametry *data* obsahující datový vektor, jež má být zapsán do paměti a parametr *addr* s hodnotou adresy, na niž mají být data zapsána. Tyto parametry jsou v těle procedury přiřazeny do stejnojmenných prvků sdílené proměnné *rom1_var* typu záznam (*record*), která umožňuje jejich vyčtení pomocí BFM. Posledním parametrem této procedury je vstupně-výstupní signál *req*. Tento signál musí být v místě volání procedury připojen na handshake signál *rom_set_req* a jeho znegování v těle procedury je tak rozpoznáno BFM *rom*.

Procedura *proc_read* slouží k okamžitému vyčtení dat z požadované adresy. Při jejím volání přijímá rovněž parametr *addr*, sloužící k určení adresy, z nichž mají být data vyčtena. Hodnota tohoto parametru je podobně jako v prvním případě uložena do stejnojmenného prvku sdílené proměnné *rom1_var* typu záznam. Druhým parametrem je výstupní signál *read_data*, do kterého je uložena hodnota prvku *data* ve sdílené proměnné typu record. Do tohoto prvku je BFM rom uloží hodnotu požadované buňky paměti při operaci čtení v procesu *p_rom_set*. Posledním parametrem je opět signál vstupně-výstupní signál req, jež musí být v místě volání procedury připojen na handshake signál *rom_set_req* a jeho znegování v těle procedury je tak rozpoznáno BFM *rom*.

Cílem bylo navrhnout paměť s možností vkládání chyb. Pro tuto funkci však není potřeba vytvářet v knihovním balíku *rom_pkg* novou proceduru. Tato funkce je řešena procedurou *proc_insert_err_val* uloženou v knihovním balíku *rom_master_pkg*. Tato procedura použije procedury knihovního balíku *rom_pkg* pro vyčtení dat z paměti, následně vyčtený vektor sečte modulo 2 s chybovým vektorem a chybná data opět zapíše do paměti.

8.4 Model řídící jednotky paměti *rom_master*

Funkce modelu *rom_master* opět vychází z řešení problémů verifikace systému načrtnutých v podkapitole 4.2 a může být shrnuta do následujících bodů:

1) Na základě volání procedur uložených v knihovním balíku *rom_master_pkg* stimulovat vstupní signál kodéru *DATA_IN* a vstupní signály paměti *ADDR* a *WE* uložené v modulu *rom*.

2) Zajistit, aby tyto signály zůstaly stabilní po dobu předstihu před nástupnou hranou hodinového signálu *CLK* a po dobu přesahu po příchodu nástupné hrany hodinového signálu *CLK*.

3) Přijímat výstupní signály z dekodéru a porovnávat jejich hodnotu s referenčními hodnotami těchto signálů, které jsou v knihovním balíku *rom_master_pkg*.

4) Kontrolovat, zda výstupní signály z dekodéru nenabývají nedefinovaných hodnot.

Vstupní a výstupní signály modelu jsou uvedeny v tabulce 7.

Tab. 7: Popis ortů modulu *rom_master*

Název portu	Typ portu	Počet bitů	Popis portu
CLK	in	1	Hlavni hodinový signál
READ_DATA	in	*n*	Výstupní signál dekodéru obsahující dekódovaná data, jež byla přečtena z paměti
ERR_DET	in	1	Příznak o detekování dvojnásobné chyby v kódovém vektoru
ERR_COR	in	1	Příznak o opravení jednonásobné chyby v kódovém vektoru
WRITE_DATA	out	*k*	Výstupní informační vektor, jež má být zapsán do paměti
ADDR	out	*m*	Číslo adresy paměti, jež má být přečtěna či přepsána
WE	out	1	Povolovací signál Write enable umožňující zápis do paměti
BFM_ERR	out	1	Signál detekujícíc chybu DUV, zjištenou některým BFM

Model obsahuje tři procesy, které zajišťují správnou funkci modelu. Prvním z nich je proces *p_x_check*, který pouze kontroluje, zda vstupní signály modelu přicházející z DUV nenabývají nedefinovaných hodnot. Kontrola nedefinovaných hodnot může být vypnuta přenastavením implicitní hodnoty generického signálu *g_X_CHECKS_ON true* na hodnotu *false*.

Druhým procesem je proces *p_rw_op* pro stimulování výstupních signálů jdoucích do kodéru a paměti. Podobně jako u BFM *rom* je zprostředkována dvojí výměna informací mezi tímto BFM a připojeným knihovním balíkem *rom_master_pkg*. První způsob výměny informací je pomocí sdílených proměnných uložených v knihovním balíku. Jejich hodnoty jsou přístupné pro BFM *rom_master*, který tak může nastavit hodnotu stimulovaných výstupních signálů na hodnotu těchto proměnných. Samotná synchronizace volání procedury a vykonávání procesu je však stejně jako u BFM *rom* řízena podle principu provozu dotaz-odpověď, tedy pomocí dvojice handshake signálů (signály pro řízení vzájemné spolupráce). V těle volané procedury je znegován signál *req*, který je napojen v místě volání procedury na první handshake signál *rw_req*. Proces *p_rw_op* je spuštěn při změně tohoto signálu. Naopak před dokončením vykonáváníní tohoto procesu je uvnitř znegován druhý handshake signál *rw_req_help*, jehož změnu rozpozná volaná procedura.

V těle procesu je nejdříve ošetřeno, aby změna výstupních signálů nebyla provedena v době času předstihu či přesahu okolo nástupné hrany hodinového signálu. Atributem ‘last_event je zjištěna doba od poslední změny hodinového signálu a vypočtena doba do nejbližšího časového okamžiku, v němž je bezpečné měnit výstupní signály. Po uplynutí této doby jsou nastaveny hodnoty výstupních signálů podle hodnot uložených v prvcích sdílené proměnné *rw_data_var* typu záznam (*record*). Podle prvku *rw_op* je rozeznáno, zda byla volána procedura pro zápis do paměti či pro čtení z paměti. V případě operace čtení je změněna pouze hodnota výstupního signálu *ADDR* na hodnotu prvku *addr* proměnné *rw_data_var*, čímž je nastavena adresa paměti, jež bude s příští nástupnou hranou hodinového signálu vyčtena. V případě operace zápisu je kromě toho také nastavena hodnota výstupního signálu *WRITE_DATA* na hodnotu prvku *data* proměnné *rw_data_var* a hodnota signálu *WE*, povolující zápis do paměti je nastavena na aktivní úroveň. Protože zápis do paměti je proveden až s nejbližší nástupnou hranou hodinového signálu *CLK*, je nutné vyčkat do této události. Zároveň s nástupnou hranou hodin jsou originální data uložena do kontrolní paměti *ctrl_mem*, vytvořené v knihovním balíku *rom_master_pkg*, jež obsahuje informace o originálních datech v každé buňce paměti. Zároveň je také vynulována informace v chybové paměti *err_mem* o počtu chyb na přepsané buňce paměti *rom*. Tato událost může proběhnout

až v okamžiku, kdy jsou data do paměti skutečně zapsána, neboť dříve by porovnáním této referenční paměti s výstupními signály z dekodéru mohlo být nesprávně označeno za chybu. Po uplynutí doby přesahu je následně signál *WE* povolující zápis opět nastaven do neaktivní hodnoty.

Třetím procesem je proces *p_sim_control*. Tento proces s každou nástupnou hranou hodinového signálu CLK porovnává, zda výstupní signály dekodéru odpovídají referenčním hodnotám těchto signálů, uchovávaných ve sdílených proměnných kontrolní paměti *ctrl_mem* a chybové paměti *err_mem*. V případě, že do buňky paměti, jejíž dekódovaná hodnota je poslána na výstup, je vložena pouze jednonásobná či žádná chyba, výstupní signál dekodéru *READ_DATA* musí odpovídat referenční hodnotě uložené v kontrolní paměti *ctrl_mem*. Výstupní signál *ERR_COR*, signalizující opravení jednonásobné chyby musí být aktivní pouze v případě, že je do paměti vložena jednonásobná chyba a výstupní signál dekodéru *ERR_DET* musí být aktivní pouze v případě, že je do paměti vložena dvojnásobná chyba. Porušení jakékoliv z těchto podmínek je pomocí signálu *BFM_ERR* oznámeno modulu *testcase*. Při vložení trojnásobné chyby, či chyby o vyšší násobnosti nejsou tyto kontroly vůbec prováděny, neboť dekodér není schopen rozeznat více než dvojnásobnou chybu.

Tato kontrola však musí být prováděna s určitým zpožděním, které je nastaveno na hodnotu *c_DECODER_COMB_DELAY* po události nástupné hrany hodinového signálu *CLK*. Události změny výstupního signálu dekodéru a porovnání tohoto signálu s referenčním signálem proběhnou v jeden simulační čas, ale v rámci rozdílných tzv. delta cyklů. Během simulace může být totiž každý okamžik simulačního času rozdělen na libovolné množství nekonečně krátkých časových okamžiků, kterým říkáme delta cykly. Každá událost, která se vykonává ve stejném simulačním čase, jako předchozí událost se vykonává v jiném delta cyklu. Až poté, co se vykonají všechny události plánované na jeden simulační čas, se tento čas posune na další časový okamžik [11]. Událost porovnání hodnot by však mohla nastat v dřívějším delta cyklu, než událost změny výstupního signálu dekodéru, což by se projevilo porušením jedné z výše uvedených podmínek a zákmitem signálu *BFM_ERR* v nulovém simulačním čase. Proto je samotné porovnání potřebné provést až v jiném simulačním čase, než probíhá změna výstupního signálu.

8.5 Pomocné moduly a knihovní balíky

Činnost systému je podporována několika pomocnými moduly a knihovními balíky Zde je uveden stručný popis vybraných objektů.

Modul osc slouží ke generování hodinového signálu o periodě, jež je definována konstantou *c_CLK_PERIOD*, která je deklarována v knihovním balíku *timing_pkg*. Generování hodin je pozastaveno po dokončení simulace voláním procedury *proc_stop_osc* z architektury modulu *testcase*.

V knihovním balíku *messages_pkg* jsou definovány procedury pro výpis hlášení o průběhu simulace. Ve verifikačním prostředí je nastavena generická konstanta *c_MSG_ON*, která povoluje výpis těchto hlášení na textový výstup simulátoru. Procedura pro výpis zprávy je nazvána *msg*, procedura pro výpis varování je nazvána *warning* a procedura pro výpis chybového oznámení je nazvána *err*. Všechny tyto procedury přijímají dva parametry a to řetězec identifikátoru modulu, v němž byla chyba objevena a řetězec krátkého popisu chyby.

9 VERIFIKACE

Jedním z cílů práce bylo provést automatickou validaci syntetizovatelných modulů, pro ověření každého vygenerovaného kodéru a dekodéru. V terminologii, používané v zadavatelské firmě ON Design Czech je pojmem validace myšleno ověření, že systém bude mít při zasazení do pracovního prostředí očekávané vstupně-výstupní chování. Validní systém tedy bude mít požadovaná data na svém výstupu. V případě modulu kodéru to znamená, že na jeho výstupu jsou generována slova z množiny kódových vektorů se správnými kontrolní bity. V případě dekodéru to znamená, že na jeho výstupu jsou správná data v případě žádné či jednonásobné chyby ve vstupních datech, v případě jednonásobné chyby ve vstupních datech je oznámeno opravení jednonásobné chyby a v případě dvojnásobné chyby ve vstupních je oznámeno detekování dvojnásobné chyby. Validací však není myšleno simulování těchto výstupních dat, ale jejich měření na reálném obvodu. Pojem validace byl nevhodně zvolen v zadání této práce, neboť zamýšleným cílem bylo provedení verifikace generovaných syntetizovatelných modulů.

Verifikace je proces, jejíž cílem je ověření, že je systém navržen správným způsobem, tedy že verifikovaný modul splňuje zadaná kritéria. Těmito kritérii může být mimo správné vstupně-výstupní chování syntetizovatelného modulu např. správná činnost simulačního modelu, podmínka maximálního zpoždění určitých signálů, požadovaná maximální délka nejdelší kombinační cesty atd.

Všechny požadavky, jež musí systém splňovat, jsou zapsány do tzv. verifikačního plánu. Součástí plánu je návrh testovacích posloupností, jež musí být vykonány, aby byl ověřen daný požadavek. Pokud jsou úspěšně vykonány všechny testovací posloupnosti, popsané u každého z požadavků, je systém označen jako úspěšně verifikovaný.

9.1 Zásady pro verifikaci

Před popisem samotného verifikačního plánu je vhodné uvést zásady pro verifikaci, jimiž se tento plán řídí:

1. Verifikační plán je rozdělen do sekcí, z nichž každá sekce popisuje verifikaci jednoho DUV (*Design Under Verification*). Pro verifikaci jednoho DUV slouží vždy jedna architektura modulu *testcase*, jejíž název je ve tvaru *tc_TESTCASE.vhd*, kde část „*TESTCASE*“ je nahrazena názvem testu. Tyto sekce jsou dále děleny na podsekce, popisující jednotlivá požadavky, jež musí DUV splňovat. Pro každý požadavek je napsána posloupnost úkonů, jež musí být vykonána pro jeho ověření. Ověření každého kritéria je popsáno jako samostatná procedura v architektuře modulu *testcase*. Každá z těchto procedur je vykonatelná samostatně, bez nutnosti předchozího úspěšného dokončení jiné procedury.

2. Pro hlášení o průběhu simulace jsou použity procedury, definované v knihovním balíku *messages_pkg*.

3. Pokud je detekována chyba DUV v průběhu vykonávání procedur modulu *testcase*, je to oznámeno pomocí chybového výpisu, obsahující krátký popis chyby. Pokud není objevena žádná chyba DUV v průběhu vykonávání procedur modulu *testcase*, je vypsáno hlášení o úspěšném dokončení.

4. DUV je verifikováno, pokud jsou splněny následující požadavky:

    a. Všechny moduly *testcase* jsou navrhnuty podle těchto zásad a všechny jsou úspěšně dokončeny (nejsou vypsána žádná chybová oznámení).

    b. Je dosaženo tzv. 100% pokrytí kódu (*code coverage*) navržených DUV. Tímto parametrem je vyjádřeno, jaká část kódu je verifikována.

    c. Jsou ověřeny všechny požadavky na jednotlivé DUV.

9.2 Verifikační plán

S ohledem na výše uvedené požadavky na verifikaci byl sestaven verifikační plán. Cílem práce je verifikovat pouze syntetizovatelné moduly kodéru a dekodéru, proto je plán rozdělen pouze do dvou sekcí, z nichž každá se věnuje verifikaci jednoho z těchto modulů. V rámci sekce verifikačního plánu jsou nejdříve krátce popsány cíle verifikace a stručně podány informace o funkci verifikačního prostředí. Zejména je připomenuto, jaké chybné chování syntetizovatelných modulů je kontrolováno verifikačním prostředím a zda je toto chování ohlašováno chybovými výpisy. Poté jsou v rámci podsekcí uvedeny konkrétní požadavky na syntetizovatelný modul a uvedena testovací posloupnost, která slouží k ověření těchto požadavků. Každá testovací posloupnost přitom ověřuje zadaný požadavek na syntetizovatelný modul bez nutnosti použít jiný syntetizovatelný modul. Sestavený verifikační plán je uveden níže:

1. **Kodér**

Popis cílů verifikace: Ověřit správnou funkcionalitu kodéru. Ověřit, že kodér generuje správné kontrolní bity.

Informace k testu: Správnost kontrolních bitů je ověřována těmito způsoby: Vstupní data kodéru jsou jím kódována a uložena na požadovanou adresu do paměti *rom1*. Vstupní data jsou také paralelně kódována pomocí funkce *encode_data* a uložena do kontrolní paměti *ctrl_mem.*

Při použití procedury *proc_read_data* volané s hodnotou *true* parametru *fast* jsou vyčtena data z paměti *rom1* mimo rozhraní a porovnána s daty v kontrolní paměti *ctrl_mem*. Pokud se tato data liší, je na textový výstup simulátoru vypsán chybový výpis.

1.1. **Ověřit správné generování kontrolních bitů.**

Požadavek: Kodér generuje správné kontrolní bity při všech možnostech vstupního informačního vektoru.

Testovací posloupnost:

1) Zapsat do paměti data, generovaná kodérem při všech možnostech vstupního informačního vektoru.

2) Po každém zápisu vyčíst data z paměti mimo rozhraní.

2. Dekodér

Popis cílů verifikace: Ověřit správnou funkcionalitu dekodéru. Ověřit, že dekodér splňuje požadavek SEC-DED.

Informace k testu: Správnost výstupních dat dekodéru je ověřována těmito způsoby: Pomocí funkce *encode_data* jsou kódovány vstupní informační bity a výsledný kódový vektor je následně uložen do paměti *rom1* a do kontrolní paměti *ctrl_mem*. Při vložení chyby do paměti je na odpovídající adresu chybové paměti *err_mem* uložena informace o násobnosti dané chyby. Do paměti Vstupní data kodéru jsou jím kódována a uložena na požadovanou adresu do paměti *rom1*. Vstupní data jsou také paralelně kódována pomocí funkce *encode_data* a uložena do kontrolní paměti *ctrl_mem.*

Při použití procedury *proc_read_data* volané s hodnotou *false* parametru *fast*, jsou data z paměti *rom1* vyčtena skrz rozhraní a dekódována pomocí modulu dekodéru. Dekodér posílá do modulu *rom_master* dekódovaná data společně s příznaky o opravení jednonásobné chyby nebo opravení dvojnásobné chyby. Pokud je násobnost chyby menší než dva ale dekódovaná data neodpovídají datům uloženým v kontrolní paměti, je na textový výstup simulátoru vypsán chybový výpis. Pokud je násobnost chyby vložené do paměti menší než tři, ale neodpovídá násobnosti detekované či opravené chyby, je na textový výstup simulátoru vypsán chybový výpis. V případě vyšší násobnosti chyby vložené do paměti dekodér nezajišťuje správné dekódování dat a chybové výpisy nejsou vypisovány.

2.1. Ověřit správné dekódování dat, která nebyla poškozena chybu

Požadavek: Při dekódování dat bez chyby odpovídají výstupní data originálním informačním bitům a není nastavena aktivní úroveň příznaků, signalizujících opravení jednonásobné chyby a detekování dvojnásobné chyby.

Testovací posloupnost:

1) Zapsat na jednu adresu paměti data generovaná funkcí *encode_data* při všech možnostech vstupního informačního vektoru.

2) Po každém zápisu vyčíst data z paměti skrze rozhraní

2.2. Ověřit správné dekódování dat, která byla poškozena jednonásobnou chybou

Požadavek: Při dekódování dat, jež byla poškozena jednonásobnou chybou, odpovídají výstupní data originálním informačním bitům a je nastavena aktivní úroveň příznaku, signalizující opravení jednonásobné chyby. Příznak signalizující detekování dvojnásobné chyby je nastaven na neaktivní úroveň.

Testovací posloupnost:

1) Zapsat na jednu adresu paměti data generovaná funkcí *encode_data* při libovolné možnosti vstupního informačního vektoru.

2) Vložit na danou adresu paměti jednonásobnou chybu.

3) Po vložení každé chyby přečíst data z adresy paměti skrze rozhraní .

4) Opakovat kroky 1 – 3 pro všechny možnosti jednonásobných chyb.

2.3. Ověřit správné dekódování dat, která byla poškozena dvojnásobnou chybou

Požadavek: Při dekódování dat, jež byla poškozena dvojnásobnou chybou, nemusí výstupní data odpovídat originálním informačním bitům, pouze je nastavena aktivní úroveň příznaku signalizující detekci dvojnásobné chyby. Příznak signalizující opravení jednonásobné chyby je nastaven na neaktivní úroveň.

1) Zapsat na jednu adresu paměti data generovaná funkcí *encode_data* při libovolné možnosti vstupního informačního vektoru.
2) Vložit na danou adresu paměti dvojnásobnou chybu.
3) Po vložení každé chyby přečíst data z adresy paměti skrze rozhraní.
4) Opakovat kroky 1 – 3 pro všechny možnosti dvojnásobných chyb.

9.3 Modul testcase

Na základě verifikačního plánu byly vytvořeny dvě architektury modulu *testcase*. Modul *testcase* je tedy tvořen entitou *tc* a architekturami *tc_coder* a *tc_decoder*, jež mohou být k entitě připojeny. Připojení architektury k entitě je provedeno v jednotce *configuration,* nacházející se v souborech, popisující jednotlivé architektury. Modul *testcase* je přitom použit jako komponenta modulu *testbench* a tak začleněn do hierarchické struktury verifikačního prostředí.

Modul *testcase* má pouze jediný vstupní signál *BFM_ERR*, který slouží k oznámení chyby verifikovaného modulu, jež byla odhalena verifikačním prostředím v průběhu simulace. Tento signál je v modulu *testbench* napojen na chybové signály všech simulačních modelů, jež mohou detekovat chybu verifikovaného modulu. Obecně může být chyba verifikovaného modulu odhalena jedním z modelů verifikačního prostředí, v proceduře knihovního balíku, který je k modelům verifikačního prostředí připojen, nebo v samotné architektuře modulu *testcase*. V prvním případě je chyba oznámena modulu *testcase* nastavením aktivní úrovně signálu *BFM_ERR* jedním ze simulačních modelů verifikačního prostředí. V druhém případě je poslána jako výstupní parametr procedury a v modulu *testcase* připojena na jeden z vnitřních signálů. Právě tímto případem je odhalení chyby v proceduře *proc_read_rom*, která je definována v knihovním balíku *rom_master_pkg*. Výstupní parametr procedury, oznamující tuto chybu je napojen na vnitřní signál modulu *testbench* pojmenovaný *read_bfm_err*.

Architektury modulu *testcase* obsahují dva procesy. Prvním je proces *p_stim_proc*, v němž jsou volány všechny procedury, kterými jsou ověřovány jednotlivé požadavky na DUV. Po úspěšném vykonání všech procedur je nastavena aktivní úroveň signálu *stimuli_done*, oznamující úspěšné dokončení stimulování všech signálů. Druhým procesem je *p_sim_control*, který je kontroluje, zda nenastala chyba syntetizovatelného modulu. Pokud modul *testcase* detekuje aktivní úroveň signálu *BFM_ERR* nebo *read_bfm_err* je simulace přerušena a na textový výstup simulátoru je vypsán chybový výpis o přerušení simulace.

Naopak pokud proces detekuje aktivní úroveň signálu *stimuli_done*, je na textový výstup simulátoru vypsán text o úspěšném dokončení simulace.

Každá architektura slouží k ověření funkčnosti jednoho syntetizovatelného modulu. Architektura *tc_coder* slouží pro verifikaci kodéru a obsahuje jedinou proceduru:

a) Procedura *proc_all_data_write_check* ověřuje zápis do paměti všech kódových vektorů, které vznikly kódováním všech možností informačních bitů. V těle této procedury jsou ve smyčce volány procedury *proc_write_data* pro zápis dat do paměti pomocí kodéru a *proc_read_rom*, pro vyčtení dat z paměti bez použití dekodéru a porovná je s referenčními daty, jež jsou z originálních dat kódována funkcí *encode_data*.

Architektura t*c_decoder* slouží pro verifikaci dekodéru a jsou ní deklarovány tři procedury:

a) Procedura *proc_all_data_write_check* ověřuje správné dekódování všech kódových vektorů, které vznikly kódováním všech možností informačních bitů. V těle této procedury jsou ve smyčce volány procedury *proc_write_data* pro zápis dat do paměti bez použití kodéru a *proc_read_rom*, pro vyčtení a dekódování z paměti a porovná je s referenčními daty, jež jsou z originálních dat kódována funkcí *encode_data*.

b) Procedura *proc_all_SE_dec_check* ověřuje správné dekódování dat, jež byla poškozena libovolnou jednonásobnou chybou. V těle procedury jsou ve smyčce volány procedury *proc_write_data* pro zápis dat do paměti bez použití kodéru, *proc_insert_err_val* pro vložení jednonásobné chyby do paměti a procedury *proc_read_rom*, pro vyčtení a dekódování z paměti a porovná je s referenčními daty, jež jsou z originálních dat kódována funkcí *encode_data*. V každé smyčce je přitom vkládána jiná jednonásobná chyba, dokud není ověřeno vložení všech možných jednonásobných chyb.

c) Procedura *proc_all_DE_dec_check* ověřuje správné dekódování dat, jež byla poškozena libovolnou dvojnásobnou chybou a funguje tak obdobně jako procedura *proc_all_SE_dec*_check, pouze jsou postupně vkládány všechny možné dvojnásobné chyby.

Vykonáním všech procedur obou architektur modulu testcase jsou verifikovány syntetizovatelné moduly kodéru a dekodéru.

9.4 Výsledky verifikace

Verifikace proběhla pomocí simulačního nástroje ModelSim. Tento simulační nástroj umožňuje mimo jiné i kontrolu tzv. pokrytí kódu (*code coverage*). Tímto parametrem je vyjádřeno, jaká část kódu DUV byla verifikací testována.

Byly verifikovány vybrané syntetizovatelné moduly kodéru a dekodéru rozšířeného Hammingova kódu o různé délce kódových slov. Výsledky testů vybraných modulů jsou uvedeny v tabulce 8.

Tab. 8: Verifikace vybraných modulů

	k	4	12	26	57
	n	8	18	32	64
Moduly generované aplikací *„Hamming_gen“*	Kodér	✓	✓	✓	✓
	Dekodér	✓	✓	✓	✓
Moduly vytvořené v projektu „UMEL_ŠB“	Kodér	-	✓	✓	-
	Dekodér	-	✓	✓	-

Vysvětlivky	
✓	Úspěšná simulace
X	Neúspěšná simulace
-	Netestováno

U všech vybraných kódů tedy proběhla verifikace úspěšně a je tedy možné předpokládat i správnou činnost aplikace v jazyku C, která generuje syntetizovatelné moduly kodéru a dekodéru. Simulátorem bylo navíc ověřeno, že verifikací bylo dosaženo 100% pokrytí kódu (*code coverage*).

ZÁVĚR

Tato práce, jež byla zadána spolupracující firmou ON Design Czech, analyzuje dostupné samoopravné kódy a na základě daných kritérií vybírá vhodný kód pro opravení jednonásobné chyby a pro detekování dvojnásobné chyby (tzv. SEC-DED) ve slově paměti typu ROM. Pro vybraný kód byla vytvořena aplikace generující syntetizovatelné moduly kodéru a dekodéru vybraného samoopravného kódu, popsané jazyku VHDL. Pro tyto moduly bylo vytvořeno verifikační prostředí, včetně modelu paměti ROM, umožňující vložení chyby do paměti. V tomto prostředí byla provedena automatická validace generovaných obou zmíněných modulů.

Teoretická část práce popisuje nezbytnou terminologii k popisu kódů a podává základy lineární algebry používané v teorii kódování. Tyto poznatky, převzaté z publikací uvedených v seznamu literatury, jsou prokládány příklady pro snazší pochopení problematiky a následně využity pro popis dvou významných skupin kódů, a to lineárních kódů a cyklických kódů. Důraz byl kladen na definování těch parametrů a vlastností kódů, z nichž vyplynula vhodnost výběru konkrétního kódu. U zmíněných skupin byly vysvětleny způsoby tvorby kódu s požadovanými vlastnostmi a naznačen způsob kódování i dekódovaní znaků. Z těchto skupin byli zvoleni čtyři zástupci, kteří jsou nejpoužívanější pro opravy chyb, a detailněji rozebrány jejich parametry a vlastnosti. Aby zvolení zástupci splňovali kritérium SEC-DED, musí mít kód minimální Hammingovu vzdálenost $d_{min} \geq 4$. Proto byly ze zvolených zástupců vyřazeny ty kódy, které tuto podmínku nesplňovaly. Výsledný výběr proto proběhl z rozšířeného Hammingova kódu a BCH kódu pro opravu dvojnásobné chyby na základě podpůrného kritéria – hodnoty informačního poměru. Na základě srovnání parametrů a vlastností těchto kódů byl jako nejvhodnější vybrán rozšířený Hammingův kód, který má oproti BCH kódu výhodnější informační poměr.

Praktická část práce nejdříve zvažuje možné metody návrhu kodéru a dekodéru a volí tu, jež je použita v projektu „UMEL_ŠB“. Zde jsou kodér a dekodér popsány parametricky v jazyku VHDL a jejich fungování je závislé pouze na změně konstant parametrů kódu a kontrolní matice v přiloženém knihovním balíku. Tento popis je univerzální, přehledný a úsporný co do rozsahu zdrojového kódu VHDL. Syntetizovatelné moduly kodéru a dekodéru v jazyku VHDL byly proto převzaty z tohoto projektu a upraveny podle požadavků ON Czech Design. Vedle úpravy převzatých modulů byla navíc vytvořena aplikace v jazyku C, která tyto moduly generuje společně s knihovním balíkem konstant. Tato aplikace řeší i optimalizaci generované kontrolní matice – minimalizaci celkového počtu logických členů pro výpočet kontrolních bitů a vyvážení počtu těchto členů pro výpočet jednotlivých kontrolních bitů. Tato optimalizace vede k úspoře plochy na čipu i zkrácení nejdelší kombinační cesty generovaných syntetizovatelných modulů kodéru a dekodéru.

K automatické verifikaci navržených syntetizovatelných modulů bylo vytvořeno verifikační prostředí popsané v jazyku VHDL. Práce popisuje způsob verifikace těchto modulů a vyvozuje požadavky na fungování jednotlivých částí verifikačního prostředí. Následně je detailně popsáno fungování prostředí jako celku, vysvětleno ověření funkčnosti jednotlivých syntetizovatelných modulů a popsáno fungování jednotlivých simulačních modelů. Součástí tohoto prostředí je i model paměti typu ROM umožňující zápis libovolné chybové hodnoty do paměti.

Poslední část práce popisuje automatickou verifikaci dat na výstupu kodéru a dekodéru. Verifikace proběhla podle standardů používaných v ON Design Czech, podle nichž

byl sestaven verifikační plán. Ten obsahuje dílčí požadavky na funkčnost systému a testovací posloupnosti, kterými je tato funkčnost ověřena. Verifikovaný modul je prohlášen za validní, pokud jsou úspěšně dokončeny všechny testovací posloupnosti verifikačního plánu. Podle verifikačního plánu byly vytvořeny testovací soubory, které řídí verifikační prostředí a jejichž vykonáním jsou data automaticky zkontrolována. Tímto způsobem byla úspěšně ověřena funkčnost vygenerovaných syntetizovatelných modulů kodéru a dekodéru pro různé počty informačních bitů, a to 4, 12, 26 a 57.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BFM	Bus funtional model
DUV	Design under verification
LSB	Least significant bit
MSB	Most significant bit
ROM	Read-only-memory
RTL	Register transfer level
SEC-DED	Single error correcting, double error detecting
UMEL_ŠB	generický projekt
n	délka kódového slova
k	počet informačních znaků
r	počet kontrolních znaků
R	informační poměr
d_{min}	minimální Hammingova vzdálenost
$GF(q)$	Galoisovo těleso s q prvky
α	prvek Galoisova tělesa
$g(x)$	generující polynom
$h(x)$	kontrolní polynom
$\mathbf{G}$	generující matice
$\mathbf{H}$	kontrolní matice

LITERATURA

[1] HLAVIČKA, J. *Číslicové systémy odolné proti poruchám*. 1. vyd. Praha: ČVUT, 1992, 330 s. ISBN 80-01-00852-5.

[2] ADÁMEK, J. *Kódování*. 1. Vydání. Praha: SNTL, 1989.

[3] HOLEŠOVSKÝ, J. *Obecné m-znakové kódy*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2010. 51 s. Vedoucí prof. RNDr. Ladislav Skula, DrSc.

[4] FROLKA, J. *BCH kódy*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 58 s. Vedoucí práce byl Ing. Jakub Šedý,

[5] HAMMING, R. W. *Error Detecting and Error Correcting Codes*. The Bell System Technical Journal. 1950, **26**(2), s. 147-160.

[6] PHELPS, A. E. *Constructing an Error Correction Code* [online]. Madison: University of Wisconsin, 2006. 7 s. [cit. 14. 12. 2015]. Dostupné z: http://pages.cs.wisc.edu/~markhill/cs552/Fall2006/handouts/ConstructingECC.pdf

[7] WALLACE, H. *Error Detection and Correction Using the BCH Code* [online]. Atlantic Quality Design, 2001 [cit. 14. 12. 2015]. Dostupné z: http://www.aqdi.com/bch.pdf

[8] ZLATOŠ, P. *Lineárna algebra a geometria: Cesta z troch rozmerov s presahmi do príbuzných odborov*. Bratislava: Marencin PT, 2011. ISBN 978-80-8114-111-9. Dostupné z: http://thales.doa.fmph.uniba.sk/zlatos/la/LAG_A4.pdf

[9] AL-ARS, Z., a A. J. VAN DE GOOR, "*Soft faults and importance of stresses in memory testing*," Design, Automation and Test in Europe Conference and Exhibition, s. 1084–1089, Feb. 2004.

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

[11] ŠŤASTNÝ, J. Simulace číslicových obvodů: úvod. *DPS Elektronika od A do Z: odborný časopis pro vývoj a výrobu v oboru elektroniky*. 2015, **4**(1), s. 23–27. ISSN 1805-5044.