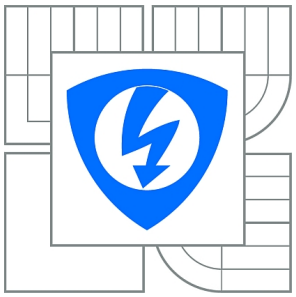
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

ZABEZPEČENÍ PŘENOSU DAT OBECNÝMI LINEÁRNÍMI BLOKOVÝMI KÓDY

DATA TRANSMITION SECURITY WITH GENERAL LINEAR BLOCK CODES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

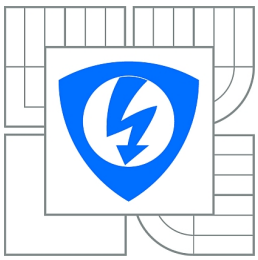
AUTOR PRÁCE — PETR DZURENDA

AUTHOR

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

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Petr Dzurenda ***ID:*** 106420
Ročník: 3 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Zabezpečení přenosu dat obecnými lineárními blokovými kódy

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte obecný lineární blokový kód, který zabezpečí přenos dat proti t = 3 nezávislým chybám, při informační rychlosti R ? 0,5. Pro tento kód vypracujte podrobný návrh realizace kodeku tohoto kódu. V návrhu realizace využijte skutečnost, že protichybový kodek bude součástí protichybového kódového systému. Ověřte funkční schopnost tohoto kodeku metodou, kterou považujete pro tento návrh za nejvhodnější.

DOPORUČENÁ LITERATURA:

[1] LEE,L.H.CH. Error-Control Block Codes for Communications Engineers. Artech House, Boston,London, ISBN 1-58053-032-X.
[2] HOUGHTON,A. Error Coding for Engineers. Kluwer academic Publishers, Boston, Dordrecht, London. 2001.
[3] ADÁMEK,J. Kódování. Nakladatelství technické literatury, Praha 1989.

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 2.6.2010

Vedoucí práce: doc. Ing. Karel Němec, CSc.

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

UPOZORNĚNÍ:

ANOTACE

Práce se v úvodu zabývá problematikou základních vlastností obecných lineárních blokových kódů, konkrétně pak vytvářením generující a kontrolní matice a způsoby dekódování obecných lineárních blokových kódů, načež jsou tyto teoretické poznatky uplatněny v další části bakalářské práce a to návrhu kódu a realizace kodeku.

Další část práce je zaměřena na konkrétní návrh obecného lineárního blokového kódu splňujícího požadavky kladené zadáním. V této kapitole lze najít způsob vytváření generující matice kódu schopného opravit tři nezávislé chyby, dále vytváření kontrolní matice pro dekódování a korekční obvod schopný tyto chyby opravit. Součástí je také popis vytvořených programů. Na konci kapitoly je uveden příklad kódování a dekódování vytvořeného kódu.

Navazující částí je realizace kodeku, kde je podrobně popsán způsob vytvoření kodéru a dekodéru lineárního blokového kódu a v následující navazující části je tento kodek odsymulován v programu Matlab Simulink.

V poslední části jsou za pomoci programu Eagle vytvořeny desky plošných spojů pro kodér a dekodér.

KLÍČOVÁ SLOVA

Lineární blokový kód, zabezpečovací kód, zabezpečení dat, samoopravný kód, kodér, dekodér

ABSTRACT

This work deals with problematic basic characteristic common linear block codes, concretely with creation generating and controlling matrix and individual ways decoding common linear block codes and then are theoretic knowledge used in the next part bachelor’s thesis.

Next part this work is bent on the concrete layout of common liner block code satisfactory specifications by submission. In this part it can by find way haw create generating matrix be able to correcting tree single errors then create control matrix for decoding and correct network for correcting errors. Programs are used in this work are explained there. In the end this part is example of coding and decoding.

Lastly work is realization codec. In this part is description of coder and decoder realizations. This codec is simulating by program Matlab Simulink in part five.

In last part are create boards of printed circuits by program Eagle for coder and decoder.

KEYWORDS

Linear block code, preventive code, data security, error-correcting code, coder, decoder

DZURENDA, P. *Zabezpečení přenosu dat obecnými lineárními blokovými kódy*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 52 s. Vedoucí semestrální práce doc. Ing. Karel Němec, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Zabezpečení přenosu dat obecnými lineárními blokovými kódy 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 jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu práce doc. Ing. Karlu Němcovi, CSc. za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne ....

(podpis autora)

OBSAH

Seznam obrázků **ix**

Seznam tabulek **x**

ÚVOD **1**

1 Zabezpečovací kódy **2**

1.1 Rozdělení zabezpečovacích kódů ... 2

1.1.1 Blokové kódy ... 2

1.1.2 Konvoluční kódy ... 4

2 Zabezpečení lineárním blokovým kódem **5**

2.1 Pojmy a způsoby zjišťování jejich vlastností ... 5

2.2 Vytvářecí matice ... 7

2.3 Kontrolní matice ... 8

2.4 Detekce chyb ... 10

2.5 Korekce chyb ... 11

2.5.1 Standardní dekódování ... 12

2.5.2 Dekódování pomocí syndromu ... 13

3 Návrh obecného lineárního blokového kódu **15**

3.1 Generující matice ... 15

3.2 Kontrolní matice ... 20

3.3 Korekce chyb ... 21

3.4 Kódování a dekódování ... 22

4 Realizace kodeku **25**

4.1 Kodér ... 25

4.2 Dekodér ... 26

4.3 Převodník [S]→[E] ... 28

4.4 Řídící obvod ... 31

5 Ověření funkčnosti **33**

6 Návrh plošného spoje **36**

7 Závěr **40**

Literatura **41**

Seznam symbolů, veličin a zkratek **42**

SEZNAM OBRÁZKŮ

Obr. 1.1: Základní rozdělení zabezpečovacích kódů 2
Obr. 1.2: Princip zabezpečení blokovým kódem 3
Obr. 1.3: Princip zabezpečení konvolučním kódem 4
Obr. 2.1: Standardní pole (Slepianovo standardní rozdělení) 12
Obr. 3.1: Vývojový diagram programu pro vytvoření generující matice 18
Obr. 3.2: Výběr vyhovující generující matice 19
Obr. 3.3: Výpis výsledků z programu prog_generujici_matice.cpp 19
Obr. 3.4: Některé možné chyby a) jedna chyba, b) dvě chyby, c) tři chyby 22
Obr. 4.1: Zapojení kodéru obecného lineárního blokového kódu (22;11) 26
Obr. 4.2: Zapojení dekodéru obecného lineárního blokového kódu (22;11) 27
Obr. 4.3: Vývojový diagram programu převodníku [S]→[E] 29
Obr. 4.4: Komprimace matice vhodná pro mikrokontroler 30
Obr. 4.5: Zapojení a průběhy řídícího obvodu a) kodéru, b) dekodéru 31
Obr. 5.1: Zapojení kodéru v programu Matlab Simulink 33
Obr. 5.2: Zapojení dekodéru v programu Matlab Simulink 34
Obr. 5.3: Blokové zapojení kodeku v programu Matlab Simulink 34
Obr. 5.4: Určení chybového slova v programu Freescale CodeWarrior 35
Obr. 6.1: Zapojení kodéru lineárního blokového kódu (22;11) – část řídící 36
Obr. 6.2: Zapojení kodéru lineárního blokového kódu (22;11) – kódovací část 37
Obr. 6.3: Zapojení dekodéru lineárního blokového kódu (22;11) – část řídící 38
Obr. 6.4: Zapojení dekodéru lineárního blokového kódu (22;11) – dekódovací část 38
Obr. 6.5: Zapojení dekodéru lineárního blokového kódu (22;11) – korekční část 39

SEZNAM TABULEK

Tab. 2.1: Standardní tabulka pro opravu jedné chyby kódu (5 ; 2) 13
Tab. 2.2: Dekódovací syndromová tabulka pro kód (7 ; 3) 13
Tab. 3.1: Tabulka korekce chybně přijatých slov 24
Tab. 4.1: Tabulka zabezpečovacích prvků 25
Tab. 4.2: Tabulka prvků syndromů 26

ÚVOD

Koncem 19. a začátkem 20. století se přenos dat stal samozřejmostí, ať už se jedná o přenos informaci přes internet, mobilní či satelitní přenos nebo jiné formy přenosu. Ze strany uživatelů, jsou pak stále více kladeny požadavky na zkvalitnění poskytovaných služeb. Za tímto účelem vznikla teorie kódování, která se zabývá zakódováním informace za určitým cílem. Jedním z hlavních cílů kódování je rychlost přenosu dat, jehož se dosahuje komprimací spočívající v odstranění nadbytečnosti (redundance) a zbytečnosti (irelevance). Dalším z hlavních cílů kódování je utajení přenášené informace z důvodu odposlechu třetích stran např. internetové bankovnictví. V neposlední řadě je v přenosu dat vyžadován bezchybný přenos, jelikož u reálných přenosových zařízení dochází k rušení, ať už se jedná o metalické či optické kabely nebo o radiový přenos, kde je toto rušení největší. Působením tohoto rušení dochází k znehodnocení informace, která je tudíž pro uživatele bezcenná. Aby se tomuto jevu předešlo, vznikly tzv. protichybové kódy.

Teorie kódování vznikla reakcí na Shannonova díla [7] z roku 1948, zabývající se touto teorií informačního přenosu. Od té doby vzniklo mnoho různých kódů, jako jsou blokové a konvoluční kódy.

Bakalářská práce se zaobírá rozebráním základních poznatků o lineárních blokových kódech, které jsou obsaženy v kapitole jedna a dvě. Další kapitola řeší již konkrétní návrh obecného lineárního blokového kódů schopného opravit tři nezávislé chyby při informační rychlosti $R \geq 0{,}5$. Kapitola čtvrtá se zaobírá realizací kodeku navrženého kódu. V páté kapitole je ověřena funkční schopnost navrženého kodeku a v poslední kapitole je uvedeno zapojení desek plošných spojů.

1 ZABEZPEČOVACÍ KÓDY

Z hlediska přenosu dat se klade velký důraz na bezchybný přenos, jelikož poškozená informace nemá pro příjemce žádnou hodnotu. Během přenosu informace dochází k vzniku chyb, které jsou způsobeny šumem. Pro zajištění bezchybného přenosu se používají tzv. zabezpečovací kódy, jejichž úkolem je detekovat a opravit vzniklé chyby. Zavedením zabezpečovacího kódu se zvýší redundance, což není příliš žádoucí, avšak dosáhne se zabezpečení přenášené informace, které je pro uživatele mnohem důležitější.

1.1 Rozdělení zabezpečovacích kódů

Zabezpečovací kódy se rozdělují do dvou skupin [6], a to na blokové a konvoluční (stromové). Tento způsob dělení spočívá na způsobu realizace zabezpečovacího procesu.

Jednotlivé skupiny se, dále rozdělují na další podskupiny viz obr. 1.1. Protože lineární blokové kódy spadají do skupiny blokových kódů, nikoli konvolučních, tak se další dělení konvolučních kódů v této bakalářské práci neuvádí.

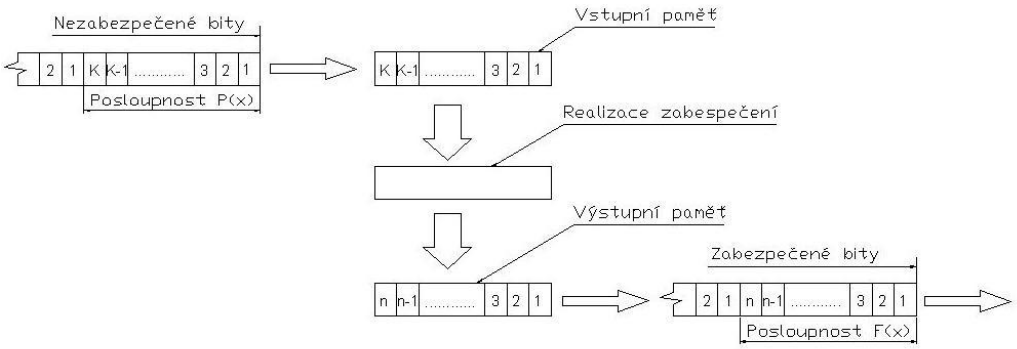
Obr. 1.1: Základní rozdělení zabezpečovacích kódů

1.1.1 Blokové kódy

Blokové kódy jsou specifické tím, že pracují s tzv. bloky [5] . Blokem se rozumí úsek oddělený od celkové posloupnosti signálových prvků. Jednotlivé úseky se nazývají kódová slova (kódová kombinace) a mají pevnou délku. Princip zabezpečení blokovým kódem vyplývá z obr. 1.2.

Na vstup kodéru se přivede polynom nezabezpečené zprávy $P(x)$ obsahující k bitů. Tyto bity se v kodéru zabezpečí a na výstupu se odesílá polynom zabezpečené zprávy $F(x)$. Tato zpráva je složena z n bitů, stávající se z původních k bitů a r bitů zabezpečovacích. Dále se lze setkat s polynomem přenesené zprávy $J(x)$, což je přenášená zabezpečená zpráva vstupující do dekodéru.

Jelikož kodér blokových kódů neobsahuje paměť zabezpečovacího procesu, tak kódování vstupního úseku je nezávislé na kódování předchozího úseku.



Obr. 1.2: Princip zabezpečení blokovým kódem

Blokové kódy se rozdělují do dvou následujících skupin [4] :

Systematické – Jak již z názvu vyplývá, uplatňuje se zde určitý systém, konkrétně se jedná o rozdělení informačních a zabezpečovacích míst. Je-li polynom zabezpečené zprávy tvořen n bity, lze jej rozdělit na k informačních bitů a r zabezpečovacích, kde $r = n - k$. Tento kód se zapisuje $(n\,;k)$. Systematické kódy se dále dělí na:

- *Lineární*: Základní myšlenkou je odvození libovolné kódové kombinace za pomoci lineární kombinace z ostatních kódových kombinací. Je to dáno zavedením algebraických operací sčítání a násobení. Lineární blokové kódy se dále dělí na další podskupiny, jejichž některé příklady jsou znázorněny na obr. 1.1,

- *Nelineární*: Jedná se o všechny systematické kódy, které nemají vlastnosti lineárních kódů.

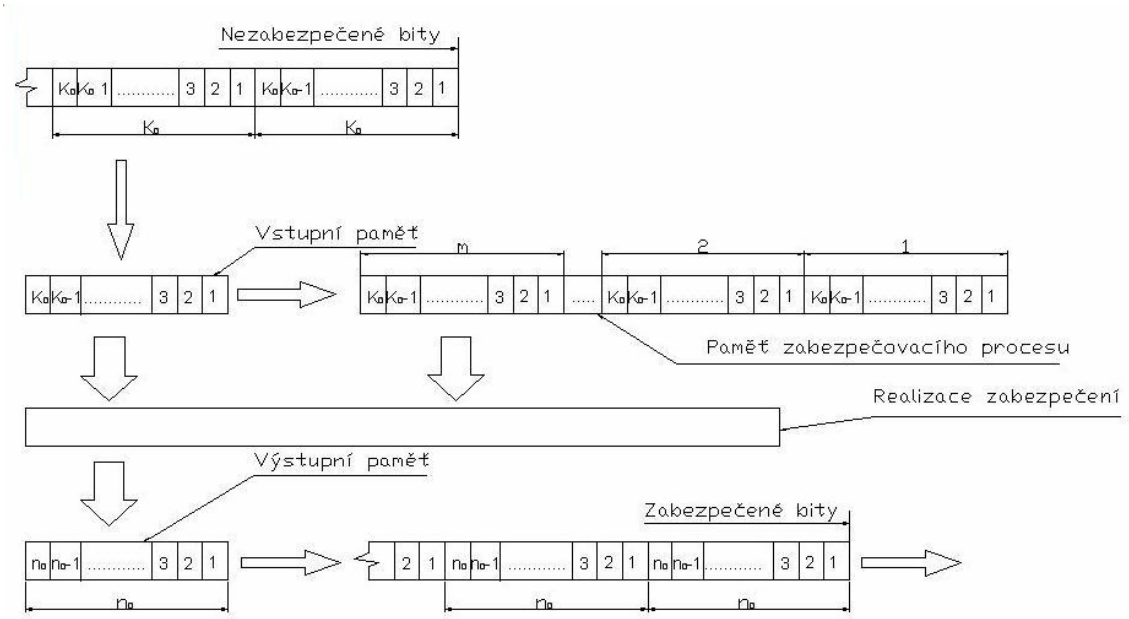
Nesystematické – Na rozdíl od systematických kódů se zde nerozděluje kódová kombinace na informační a zabezpečovací část. Zabezpečení spočívá v umístění nul a jedniček v kódové kombinaci. Tyto kódy se dále dělí na:

- *Kódy s vnitřním zákonem*: Jednotlivé kódové kombinace se sestavují podle různých předem definovaných pravidel např. tzv. Izokódy (každá kódová kombinace obsahuje stejný počet nul a jedniček), Bergerův kód (dvojkový kód) či Reed-Müllerův kód (opravující t násobné chyby).

- *Kódy bez vnitřního zákona*: Jsou to takové kódy, u kterých lze správnost kódové kombinace ověřit jen za pomoci dekódovací tabulky.

1.1.2 Konvoluční kódy

Název konvolučních (stromových) kódů [5] je dán způsobem kódovacího procesu, pro jehož vyjádření se nejčastěji využívá graf typu strom. Princip kodéru konvolučních kódů spočívá na přiřazení n_0 zabezpečených znaků ke k_0 informačním znakům, přičemž nezáleží jen na konkrétní k_0-tici prvků na vstupu, ale i na několika k_0-tic předchozích. Tento jev je dán tím, že konvoluční kodér na rozdíl od blokového obsahuje paměť zabezpečovacího procesu, která obsahuje m-násobek k_0 bitů. Z této paměti a vstupní paměti se v bloku realizace zabezpečení vytvoří zabezpečená posloupnost, tvořená úseky zabezpečené zprávy n_0, jak lze vidět na obr. 1.3. Další dělení konvolučních kódů zde není uvedeno, jelikož to není předmětem této bakalářské práce.



Obr. 1.3: Princip zabezpečení konvolučním kódem

2 ZABEZPEČENÍ LINEÁRNÍM BLOKOVÝM KÓDEM

Z popisu a rozboru uvedeném v kapitole jedna, vyplývá, že lineární blokové kódy jsou podskupinou systematických kódů, což znamená, že obsahují k informačních a r zabezpečovacích bitů a zapisují je jako kódy $(n;k)$. V praxi jsou používány dva základní způsoby zadávání lineárních blokových kódů [4] :

- Pomocí vytvářecí matice $[G]$: Tento způsob využívají všechny lineární blokové kódy a jimi se také bude zabývat tato kapitola.

- Pomocí vytvářecího mnohočlenu $G(x)$: Tento způsob využívají cyklické kódy, tvořící podskupinu lineárních kódů.

2.1 Pojmy a způsoby zjišťování jejich vlastností

V této kapitole jsou uvedeny pouze některé pojmy lineárních blokových kódů [8] související s touto bakalářskou prací a pro lepší porozumění pojmů jsou uvedeny i způsoby zjištění jednotlivých vlastností.

Hammingova vzdálenost: Je celkový počet odlišných znaků dvou a více slov. Hammingova vzdálenost se označuje d. Jsou-li dána dvě slova $p_1p_2...p_n$ a $f_1f_2...f_n$, tak Hammingova vzdálenost je počet odlišných znaků těchto slov.

$p_1p_2...p_7$: 1 1 0 0 1 1 0 $\Rightarrow$ $d=3$

$f_1f_2...f_7$: 1 1 1 0 0 1 1

Nejmenší Hammingova vzdálenost: Je nejmenší Hammningova vzdálenost ze všech Hammnigových vzdáleností mezi jednotlivými kódovými slovy. Nejmenší Hammningovu vzdálenost se označuje $d_{\min}$.

g_1 = 1 1 1 0 0 1 $\Rightarrow$ $d_{12}=4$, $d_{13}=3$

g_2 = 0 0 1 1 0 0 $\Rightarrow$ $d_{23}=5$ $\Rightarrow d_{\min}=\min\{d\}=3$

g_3 = 0 1 0 0 1 1

Hammingova váha: Je počet nenulových znaků v kódovém slově. Hammingova váha se značí w.

1 1 0 0 1 1 0 $\Rightarrow$ $w=4$

Detekční schopnost kódu: Schopnost kódu najít (detekovat) chybu. Je-li dán kód s $d_{\min}$, pak je tento kód schopen najít t_d chyb. Platí, že

$$d_{\min} \geq t_d + 1. \tag{2.1}$$

Korekční schopnost kódu: Schopnost kódu opravit chybu. Je-li dán kód s $d_{\min}$, pak je schopen opravit právě t_k chyb. Platí, že

$$d_{\min} \geq 2 \cdot t_k + 1. \tag{2.2}$$

Detekční a korekční schopnost kódu: Schopnost kódu najít (detekovat) a opravit chybu. Je-li dán kód s $d_{\min}$, pak je tento kód schopen detekovat t_d a opravit t_k chyb. Platí, že

$$d_{\min} \geq t_d + t_k + 1. \tag{2.3}$$

Plotkinova hranice: Používá se k zjištění minimální potřebné délky slova, za potřebné detekční nebo korekční schopnosti kódu. Pro lineární blokové kódy platí

$$n = k + r, \tag{2.4}$$

kde n se vypočte dle vztahu

$$n \geq \frac{F \cdot d_{\min} - 1}{F - 1}. \tag{2.5}$$

Pro binární kód je $F = 2$, tudíž upravený vztah má tvar

$$n \geq 2 \cdot d_{\min} - 1. \tag{2.6}$$

Počet zabezpečovacích znaků je dán rovnicí

$$r \geq \frac{F \cdot d_{\min} - 1}{F - 1} - (1 + \log_F d_{\min}), \tag{2.7}$$

kde po úpravě opět vzniká vztah pro binární kód

$$r \geq 2 \cdot d_{\min} - \log_2 d_{\min} - 2. \tag{2.8}$$

Lze se setkat i s jinými způsoby určení generující matice s potřebnou detekční či korekční schopností např. Hammingova hranice nebo Gilbertova hranice [2].

2.2 Vytvářecí matice

Na vstup kodéru je přivedena sekvence k znaků tzv. informačních. Tyto znaky jsou označeny vektorem $[P]=[p_1 p_2 ... p_k]$. Kodér zabezpečí informační bity pomocí redundantních zabezpečovacích r bitů a na jeho výstupu je vyslán vektor $[F]=[f_1 f_2 ... f_n]$.

U lineárních blokových kódů mohou nastat dva extrémní případy:

1) Je-li $k=0$, pak nedochází k přenášení informace.
2) Je-li $k=n$, pak dochází k tomu, že kódové slovo neobsahuje redundanci a tudíž není možné informaci zabezpečit. Jedna chyba je přenesena z jednoho kódového slova na druhé a tudíž ji není možné detekovat.

Lineární blokové kódy jsou definovány k lineárně nezávislými kódovými slovy $G_1 G_2 ... G_k$. Pak je výstupní vektor dán rovnicí

$$F = p_1 G_1 + p_2 G_2 + ... + p_{k-1} G_{k-1} + p_k G_k . \tag{2.9}$$

Napíší-li se tyto kódová slova pod sebe, vznikne generující matice $[G]$, stávající se z k řádků a n sloupců, přičemž žádný řádek není lineární kombinací ostatních. Generující matice má tvar

$$[G]=\begin{bmatrix} G_1 \\ G_2 \\ : \\ : \\ G_k \end{bmatrix}=\begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ : & : & : & : \\ : & : & : & : \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix}, \tag{2.10}$$

kde $g_{i1}, g_{i2} ... g_{in}$ jsou q-rozměrné symboly, platí $0 \le i \le k$.

Každá vytvářecí matice musí splňovat následující body [3]:

- jednotlivé řádky jsou tvořeny kódovými slovy,
- kódová slova jsou lineární kombinací řádků,
- jednotlivé řádky nejsou lineárně závislé.

Je-li dána vytvářecí matice $[G]$, pak lze pomocí ní zabezpečit příchozí zprávu. Zabezpečení se provádí tak, že z celkové nezabezpečené zprávy se vybere k bitů, označených jako vektor nezabezpečené zprávy $[P]$. Tento vektor se vynásobí vytvářecí maticí a tím vznikne vektor zabezpečené zprávy $[F]$. Platí rovnice

$$[F]=[P]\times[G], \tag{2.11}$$

které odpovída zápis

$$\begin{bmatrix} p_1 & p_2 & \dots & p_k \end{bmatrix} \times \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ : & : & : & : \\ : & : & : & : \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{bmatrix} = \begin{bmatrix} f_1 & f_2 & \dots & f_n \end{bmatrix}. \tag{2.12}$$

Pro zakódování dat lineárním blokovým kódem je důležité, aby byla generující matice $[G]$ v systematickém tvaru

$$[G] = [E \mid B], \tag{2.13}$$

kde E je jednotková matice k-tého řádu. Pokud tak není, může se každý lineární kód, který není systematický na systematický převést pomocí permutace jednotlivých znaků v kódovém slově [1], [3].

$$[G] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \Leftrightarrow [G'] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad [G_1'] = [E \mid B] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

2.3 Kontrolní matice

Pomocí matice $[G]$ z kapitoly 2.2 jsou data zakódována a tím je získán vektor zabezpečené zprávy $[F]$, který byl vyslán na telekomunikační kanál. Na straně příjemce je potřeba zkontrolovat bezchybnost přenosu a právě za tímto účelem se používá tzv. kontrolní matice $[H]$. Tato matice obsahuje r řádků a n sloupců. Pro transponovanou kontrolní matici platí vztahy:

$$[G] \times [H]^T = 0, \tag{2.14}$$

$$[F] \times [H]^T = 0, \tag{2.15}$$

kde H^T je transponovaná matice H. Jsou-li tyto dvě podmínky splněny, pak se jedná o kontrolní matici lineárního blokového kódu. Kontrolní matice je odvozena od generující matice a má tvar

$$[H]=\begin{bmatrix} H_1 \\ H_2 \\ : \\ : \\ H_r \end{bmatrix}=\begin{bmatrix} h_{11} & h_{12} & ... & h_{1n} \\ h_{21} & h_{22} & ... & h_{2n} \\ : & : & : & : \\ : & : & : & : \\ h_{r1} & h_{r2} & ... & h_{rn} \end{bmatrix}, \tag{2.16}$$

$h_{i1}, h_{i2}, ..., h_{in}$ jsou q-rozměrné symboly, platí $0 \le i \le r$.

Pro vytvoření kontrolní matice, je důležité, aby byla generující matice v systematickém tvaru [1]. Není-li tomu tak, je potřeba danou matici upravit viz. předchozí kapitola. Získáním systematické matice vzniká matice skládající se ze dvou podmatic:

- *Podmatice* E: Jedná se o jednotkovou matici (informační matici), která je dána jedničkami vyskytujícími se pouze v hlavní diagonále. Tato matice má rozměr $k \times k$.
- *Podmatice* B: Jedná se o tzv. zabezpečovací podmatici, která má k řádků a r sloupců.

Pomocí těchto dvou podmatic se sestaví kontrolní matice $[H]$, která se bude také skládat ze dvou podmatic. Matice $[H]$ bude ve tvaru:

$$[H]=[B^T \mid E], \tag{2.17}$$

kde jednotlivé matice jsou:

- *Podmatice* B^T: Jedná se o transponovanou podmatici B, tudíž se stává z r řádků a k sloupců.
- *Podmatice* E: Jedná se opět o jednotkovou matici, tentokrát má však rozměr $r \times r$.[1][4]

Jeli tedy dána generující matice

$$[G]=\begin{bmatrix} 1 & 0 & ... & 0 & p_{11} & p_{12} & ... & p_{1r} \\ 0 & 1 & ... & 0 & p_{21} & p_{22} & ... & p_{2r} \\ : & : & : & : & : & : & : & : \\ : & : & : & : & : & : & : & : \\ 0 & 0 & ... & 1 & p_{k1} & p_{k2} & ... & p_{kr} \end{bmatrix} = \textit{např.} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix},$$

potom má kontrolní matice tvar

$$[H]=\begin{bmatrix} p_{11} & p_{21} & ... & p_{k1} & 1 & 0 & ... & 0 \\ p_{12} & p_{22} & ... & p_{k2} & 0 & 1 & ... & 0 \\ : & : & : & : & : & : & : & : \\ : & : & : & : & : & : & : & : \\ p_{1r} & p_{2r} & ... & p_{kr} & 0 & 0 & ... & 1 \end{bmatrix} = \textit{např.} \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

2.4 Detekce chyb

Během přenosu informace telekomunikačním kanálem může dojít k vzniku chyby, tj. vyšle-li se zabezpečený vektor $[F]$ a přijme-li se vektor $[J]$, vzniká chybový vektor $[E]$ daný jejich rozdílem. Platí tedy, že pokud se přičte k vyslanému slovu chybový vektor, vzniká přijaté slovo. Platí tedy rovnice

$$[J] = [F] + [E] \ . \tag{2.18}$$

Z uvedeného vyplývá, že jedničky obsažené v chybovém vektoru znamenají chybu.

Pro lineární blokové kódy platí, že minimální Hammingova vzdálenost je rovna minimální Hammningově váze nenulového kódového slova. Pak lineární blokové kódy opravují t-násobné chyby, když všechna kódová slova mají minimální Hammingovu váhu větší než t.

K detekci chyb se používá tzv. syndrom. Syndrom se označuje jako $[S]$ a je určen pomocí kontrolní matice $[H]$. Princip spočívá v tom, že se přijaté slovo vynásobí inverzní kontrolní maticí a výsledným součinem je vektor, který se nazývá syndrom. Platí tedy že

$$[S] = [J] \times [H^T] . \tag{2.19}$$

Je-li syndrom nulový, pak se předpokládá, že došlo k bezchybnému přenosu, viz níže.

$$[1\ 0\ 0\ 1\ 1\ 1\ 0] \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [0\ 0\ 0]$$

Toto pravidlo ovšem není 100% -ní! Platí pravidlo, že chybové slovo nesmí být stejné jako slovo kódové. Pokud to nastane, pak chybu nelze najít.

Příklad

Je vyslané slovo $[F] = [1\ 0\ 0\ 1\ 1\ 1\ 0]$. Během přenosu vznikne vlivem šumu chybové slovo $[E] = [0\ 1\ 1\ 1\ 0\ 1\ 0]$, které je zároveň kódovým slovem. Na vstupu dekodéru se tedy přijme slovo, které je součtem těchto dvou slov.

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1\ 1\ 0 \\ +\ 0\ 1\ 1\ 1\ 0\ 1\ 0 \\ \hline 1\ 1\ 1\ 0\ 1\ 0\ 0 \end{array}$$

Přijaté slovo není 1001110 ale 1110100, tudíž evidentně došlo k chybě.

$$[1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0]\times\begin{bmatrix}1&0&1\\1&1&1\\1&1&0\\0&1&1\\1&0&0\\0&1&0\\0&0&1\end{bmatrix}=[0\ \ 0\ \ 0].$$

Je vidět, že i přesto, že došlo k chybnému přenosu je syndrom nulový a tudíž by se mělo jednat o bezchybný přenos.

V případě, že syndrom není nulový tj. $[S]\neq 0$, dochází k chybě vždy.

$$[1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0]\times\begin{bmatrix}1&0&1\\1&1&1\\1&1&0\\0&1&1\\1&0&0\\0&1&0\\0&0&1\end{bmatrix}=[1\ \ 0\ \ 0].$$

Z uvedeného vyplývá, že došlo k chybě přenosu zprávy. Detekční kódy jsou schopny pouze chybu detekovat. Vyjde-li u nich syndrom nenulový, pak nedochází k opravě, ale k žádosti o opětovné vyslání daného slova [3].

2.5 Korekce chyb

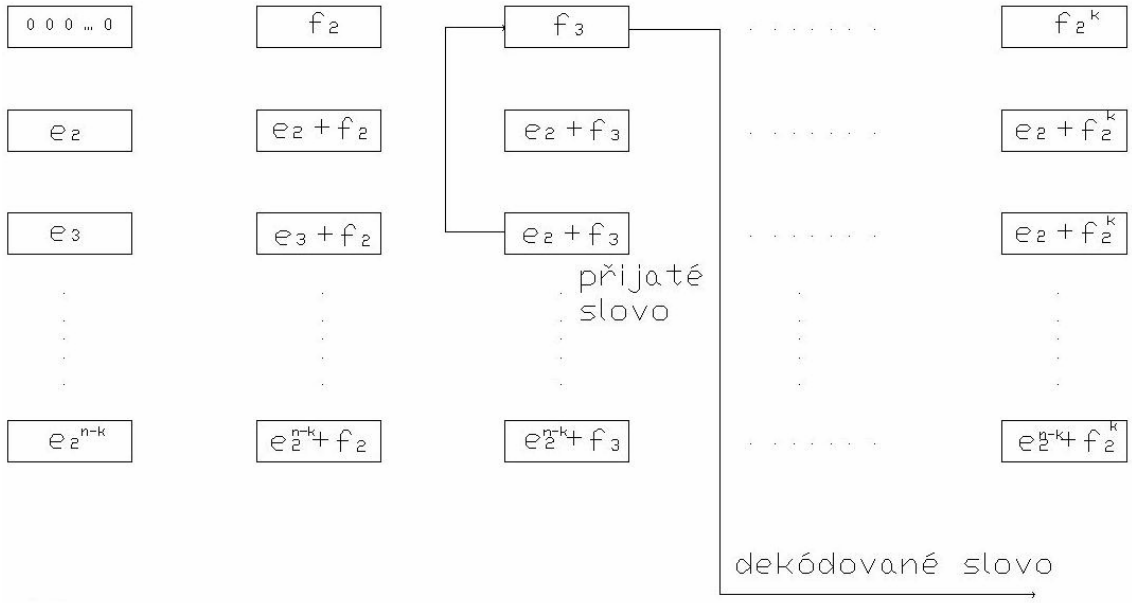
Při opravování chyb lze stejně jako v případě objevování chyb zjistit jaký druh chyby daný dekodér dokáže opravit. Dekódováním se rozumí předpis δ přiřazující kódové slovo $\delta([F])$ přijatému slovu $[J]$. Aby došlo k správnému dekódování musí platit, že dojde-li k ovlivnění vyslaného slova $[F]$ chybovým slovem $[E]$, pak

$$\delta([F]+[E])=[J]\ . \tag{2.20}$$

Je-li tato podmínka splněna, pak při dekódování lineární blokový kód opravuje přijatá slova. Rozlišují se základní dva druhy dekódování [1], [3] a to standardní a syndromové.

2.5.1 Standardní dekódování

Standardní dekódování [3] se provádí pomocí tzv. standardního pole lineárního blokového kódu nebo také Slepianovo standardní rozmístění viz. obr. 2.1.



Obr. 2.1: Standardní pole (Slepianovo standardní rozdělení)

Jedná se o tabulku, kde jednotlivé řádky jsou třídy a na první pozici každého řádku je tzv. reprezentant třídy, což je vybrané slovo s nejmenší Hammingovou váhou. Princip spočívá ve vytvoření tohoto standardního pole, které má q^{n-k} řádků a q^k sloupců. U binárních kódů se za q dosazuje dvojka. V dalším kroku se vyplní potřebné hodnoty. Reprezentant první třídy je vždy 000…0 jelikož se jedná o slovo s nejnižší váhou. Za tímto slovem následují kódová slova. Na druhém řádku se volí jako reprezentační slovo další slovo s nejmenší Hammingovou vahou, které není obsaženo na předchozích řádcích. Následující slova jsou dány součtem reprezentanta a kódového slova.

Tímto způsobem se vyplní celé standardní pole. Obdrží-li se poté příchozí slovo, které je obsaženo v nějaké třídě, pak opraveným dekódovaným slovem je slovo obsažené v tomtéž sloupci a zároveň v první třídě.

Z uvedeného vyplývá, že standardní dekódování opravuje pouze ta chybová slova, která jsou nadefinována jako reprezentanty jednotlivých tříd.

Příklad

Je-li dán kód $(5\,;2)$ opravující jen jednu chybu s generující maticí

$$[G]=\begin{bmatrix}1&0&0&1&1\\0&1&1&1&1\end{bmatrix}.$$

Kódová slova jsou 00000, 10011, 01111 a 11100 ($10011 \oplus 01111$), což odpovídá $q^k = 2^2 = 4$. Pole má mít $q^{n-k} = q^{5-2} = 8$ řádků, tudíž jednotlivými reprezentanty jsou slova 00000, 00001, 00010, 00100, 01000, 10000, 00101 a 00110. Jelikož se však jedná

o korekci jen jedné chyby, budou tedy správně opravena jen ty přijatá slova, která obsahují chybové vektory 00000, 00001, 00010, 00100, 01000, 10000.

Bylo-li vysláno slovo $[F]=[0\,1\,1\,1\,1]$ a na tento vektor bude působit chybové slovo $[E]=[0\,0\,0\,1\,0]$, pak získané slovo je $[J]=[0\,1\,1\,0\,1]$ a tomuto slovu se vyhledá příslušné opravené slovo $[R]=[0\,1\,1\,1\,1]$, jak již vyplývá z tab. 2.1.

Tab. 2.1: Standardní tabulka pro opravu jedné chyby kódu (5 ; 2)

00000	01111	10011	11100
00001	01110	10010	11101
00010	01101	10001	11110
00100	01011	10111	11000
01000	00111	11011	10100
10000	11111	00011	01100

Mezi základní výhodu standardního dekódování patří, že žádný jiný kód neopravuje větší množství chyb. Jeho hlavní nevýhodou je však velmi pomalé dekódování.

2.5.2 Dekódování pomocí syndromu

Standardní dekódování popsané v předcházející kapitole je velmi pomalé, jelikož se musí procházet 20 slov v poli při každém přijetí nového slova $[J]$. Běžně se však používají lineární blokové kódy, které mají délku 62 a potom by se muselo procházet 2^{62} slov, což je cca $4{,}6\cdot 10^{18}$. Tuto nevýhodu odstraňuje dekódování, které využívá kontrolní matici $[H]$ daného kódu a syndrom.

Bylo zjištěno, že slova ležící v jedné třídě mají stejný syndrom. Kvůli tomuto zjištění se mnohonásobně zmenší dekódovací tabulka, protože stačí znát reprezentanta a syndrom viz. tab. 2.2.

Tab. 2.2: Dekódovací syndromová tabulka pro kód (7 ; 3)

Chybový vektor	Syndrom
0000001	0001
0000010	0010
0000100	0100
0001000	1000
0010000	0111
0100000	1110
1000000	1011

Je důležité si uvědomit, že přijaté slovo má stejný syndrom jako chybové slovo, jelikož platí

$$[S] = [J] \times [H^T] = ([F] + [E]) \times [H]^T = [F] \times [H^T] + [E] \times [H^T]. \qquad (2.21)$$

Odsud potom plyne, že

$$[S] = [E] \times [H]^T \quad \Rightarrow \quad [J] \times [H]^T = [E] \times [H]^T. \qquad (2.22)$$

Jednotlivé syndromy se vypočítají dle vztahu 2.22.

Přijme-li se slovo $[J]$, tak se zjistí jeho syndrom a pokud bude nenulový, pak se určí z tabulky příslušné chybové slovo $[E]$. Je-li známo chybové slovo, není již problém opravit vzniklou chybu dle vztahu 2.20.

Příklad

Vyslané slovo je $[F] = [1\,0\,1\,1\,1\,0\,0]$, přijaté je slovo $[J] = [1\,0\,0\,1\,1\,0\,0]$. Kód opravuje pouze jednu chybu a má transponovanou kontrolní matici ve tvaru

$$[H^T] = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Jednotlivé reprezentanty jsou 0000001, 0000010, 0000100, 0001000, 0010000, 0100000 a 1000000 viz. tab. 2.2. Syndrom přijatého slova je

$$[1\ 0\ 0\ 1\ 1\ 0\ 0] \times \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [0\ 1\ 1\ 1].$$

Z tabulky vyplývá, že danému syndromu odpovídá chybové slovo $[E] = [0\,0\,1\,0\,0\,0\,0]$. Odečtením chybového slova od přijatého slova se získá opravené slovo stejné hodnoty jako slovo vyslané tj. $[R] = [1\,0\,1\,1\,1\,0\,0]$.

3 NÁVRH OBECNÉHO LINEÁRNÍHO BLOKOVÉHO KÓDU

Tato kapitola se zabývá vytvořením obecného lineárního blokového kódu (dále jen kódu) tak, aby splňoval úkol obsažený v zadání. Z důvodu náročnosti a zdlouhavosti některých výpočtů bylo využito vyššího programovacího jazyka C++, konkrétně programovacího prostředí C++ Builder 6.

Zadání vyžaduje návrh kódu schopného zabezpečit přenos digitálních dat proti $t = 3\,\text{bity}$ nezávislým chybám při informační rychlosti $R \geq 0{,}5$. Z těchto dvou předpokladů se vychází při návrhu zabezpečení.

Z předchozích kapitol je zřejmé, že má-li mít kód určité korekční schopnosti, musí splňovat rovnici 2.2. Pro korekční schopnost lineárního blokového kódu schopného opravit tři chyby se minimální Hammingova vzdálenost vypočítá dle vztahu 2.2

$$d_{\min} \geq 2 \cdot t_k + 1,$$

$$d_{\min} \geq 2 \cdot 3 + 1,$$

$$d_{\min} \geq 7.$$

Dále pro Hammingovu váhu platí

$$d_{\min} = \min\{w\},$$

$$w \geq d_{\min},$$

$$w \geq 7.$$

3.1 Generující matice

Pro vytvoření generující matice je potřeba znát počet zabezpečovacích r, informačních k nebo maximální délku slova n a dále vlastnosti generované matice k opravení daného množství chyb. V tomto případě tyto vlastnosti udávají parametry $d_{\min}$ a w, které již byly vypočteny. K vytvoření generující matice se zde využije tzv. Plotkinovy hranice.

1) Minimální délka tohoto kódu je podle vztahu 2.6

$$n \geq 2 \cdot d_{\min} - 1,$$

$$n \geq 2 \cdot 7 - 1,$$

$$n \geq 13 .$$

2) Množství minimálního počtu zabezpečovacích znaků vychází ze vztahu 2.8

$$r \geq 2 \cdot d_{\min} - \log_2 d_{\min} - 2 ,$$

$$r \geq 2 \cdot 7 - \log_2 7 - 2 ,$$

$$r \geq 10 .$$

Je-li znám počet zabezpečovacích znaků r a celková délka kódu n, pak se počet zabezpečovaných znaků k vypočte dle rovnice 2.4 následovně

$$k = n - r = 13 - 10 = 3 .$$

Nyní jsou vypočteny základní parametry kódu schopného opravit $t = 3$ bitů nezávislým chybám jak je požadováno v zadání. Dalším požadavkem je, aby toto zabezpečení probíhalo při informační rychlosti $R \geq 0{,}5$. Informační rychlost je dána podílem informačních bitů k celkové délce kódu a pro tento kód vychází tedy

$$R \geq \frac{k}{n},$$

$$R \geq \frac{3}{13} = 0{,}23 .$$

Je vidět, že navržený kód nedisponuje požadovanou informační rychlostí a proto je potřeba jej poopravit. Aby zde bylo dosaženo požadované informační rychlosti, je potřeba zvýšit počet informačních bitů na $k = 11$, čímž se zvýší také celková délka kódu na $n = 22$. Počet zabezpečovacích bitů je potřeba zvýšit o jeden tj. $r = 11$. Výsledná informační rychlost po úpravě parametrů vychází

$$R \geq \frac{11}{22} = 0{,}5 ,$$

čímž jsou splněný požadavky kladené zadáním.

Nyní lze sestrojit generující matici, která se skládá z jednotkové podmatice a z kontrolní podmatice o rozměru 11×11, viz níže.

$$[G]=\left[\begin{array}{c|c} \begin{matrix}1 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 0\\ 0 & \dots & 1\end{matrix} & \begin{matrix}p_{11} & p_{12} & p_{13} & p_{14} & p_{15} & p_{16} & p_{17} & p_{18} & p_{19} & p_{110} & p_{111}\\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} & p_{26} & p_{27} & p_{28} & p_{29} & p_{210} & p_{211}\\ p_{31} & p_{32} & p_{33} & p_{34} & p_{35} & p_{36} & p_{37} & p_{38} & p_{39} & p_{310} & p_{311}\\ p_{41} & p_{42} & p_{43} & p_{44} & p_{45} & p_{46} & p_{47} & p_{48} & p_{49} & p_{410} & p_{411}\\ p_{51} & p_{52} & p_{53} & p_{54} & p_{55} & p_{56} & p_{57} & p_{58} & p_{59} & p_{510} & p_{511}\\ p_{61} & p_{62} & p_{63} & p_{64} & p_{65} & p_{66} & p_{67} & p_{68} & p_{69} & p_{610} & p_{611}\\ p_{71} & p_{72} & p_{73} & p_{74} & p_{75} & p_{76} & p_{77} & p_{78} & p_{79} & p_{710} & p_{711}\\ p_{81} & p_{82} & p_{83} & p_{84} & p_{85} & p_{86} & p_{87} & p_{88} & p_{89} & p_{810} & p_{811}\\ p_{91} & p_{92} & p_{93} & p_{94} & p_{95} & p_{96} & p_{97} & p_{98} & p_{99} & p_{910} & p_{911}\\ p_{101} & p_{102} & p_{103} & p_{104} & p_{105} & p_{106} & p_{107} & p_{108} & p_{109} & p_{1010} & p_{1011}\\ p_{111} & p_{112} & p_{113} & p_{114} & p_{115} & p_{116} & p_{117} & p_{118} & p_{119} & p_{1110} & p_{1111}\end{matrix} \end{array}\right]$$

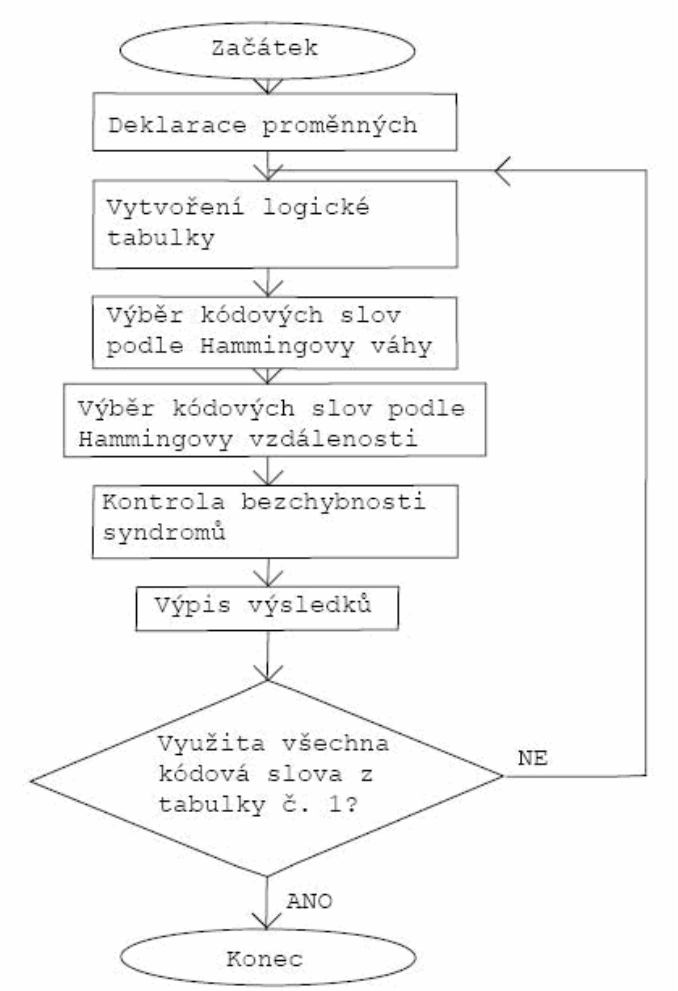
Z uvedené matice je zřejmé, že se skládá z jednotkové podmatice, pro kterou platí, že minimální Hamminhova vzdálenost je $d_{\min}=2$ a Hammingova váha je $w=1$. Z tohoto předpokladu a z dřívějších zjištění minimální Hammingovy vzdálenosti a váhy platících pro celou generující matici lze odvodit, že zabezpečovací podmatice je taková matice, která má minimální Hammingovu vzdálenost větší nebo rovno pěti a zároveň je její Hammingova váha větší nebo rovno šesti. Musí se tedy nalézt taková slova délky 11, která tuto podmínku splňují. Za tímto účelem byl použit program prog_Generujici_matice.cpp naprogramovaný v jazyce C++, jehož vývojový diagram je uveden na obr. 3.1.

Na začátku programu se deklarují proměnné potřebné pro běh programu mezi něž patří mimo jiné počet zabezpečovacích znaků `r`, celková délka kódu `N`, minimální Hammingova vzdálenost `vzdalenost`, Hammingova váha `vaha` a předvyplněná kontrolní matice `kontrol`. Dále je zde nadefinována matice chybových slov `matE` a matice syndromů `matS`.

Po deklaraci proměnných nastává vytvoření logické tabulky obsahující $2^r=2^{11}=2048$ řádků, kde první řádek je tvořen samými nulami a poslední samými jedničkami.

Jakmile je vytvořena logická tabulka, tak dochází k výběru kódových slov podle Hammingovy váhy. V tomto případě jsou vybrána ta kódová slova, která mají Hammingovu váhu větší nebo rovnu šesti.

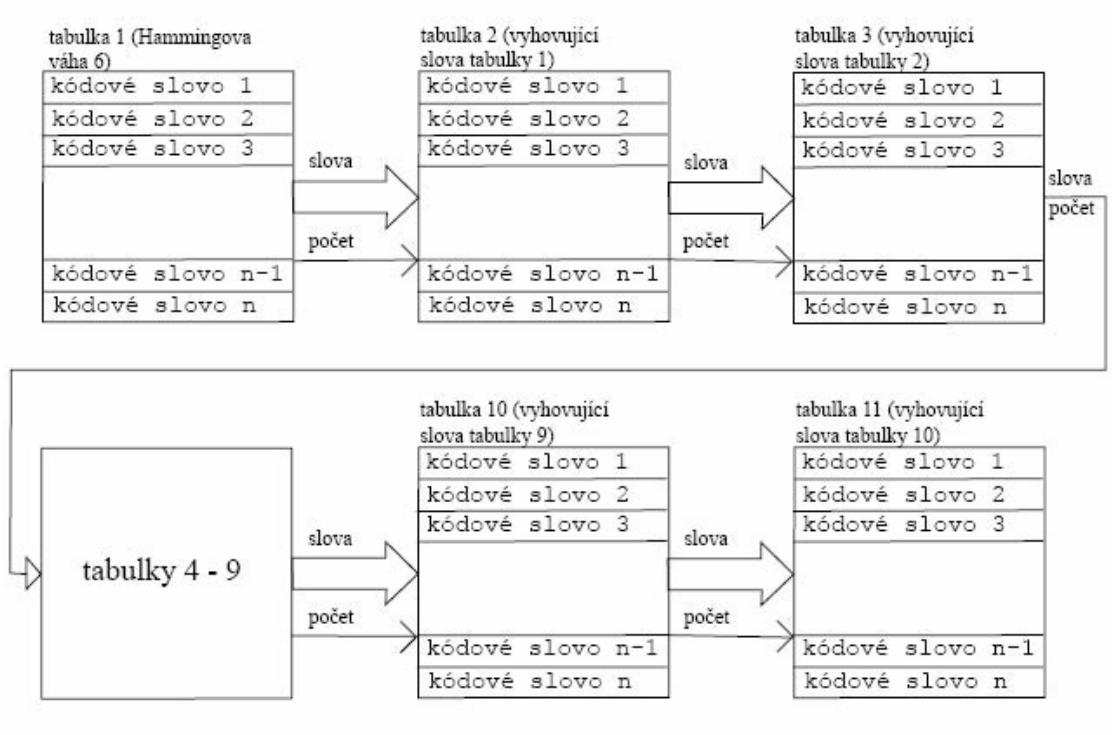
Další částí programu je výběr slov, které mají minimální Hammingovu vzdálenost větší nebo rovnu pěti. Při návrhu kódu bylo zjištěno, že pokud byl kód navržen z původních výsledků Plotkynovy hranice tj. kód by měl parametry $n=13$, $k=3$ a $r=10$, bylo by pro zjištění generující matice dostačující zjistit kódová slova s Hammingovou vzdáleností větší nebo rovno pěti odvozená od prvního kódového slova tj. slovo začínající šesti jedničkami a další kódová slova by se určovala z předchozího slova stejným způsobem.



Obr. 3.1: Vývojový diagram programu pro vytvoření generující matice

Jelikož byl počet informačních znaků zvýšen o osm, došlo k tomu, že výsledná generující matice obsahovala v zabezpečovací části kódová slova na jednotlivých řádcích splňující minimální Hammingovu vzdálenost, avšak nebyla splněna pro všechny možné kombinace kódových slov. K vyřešení tohoto problému je vytvořeno deset cyklů které vybírají jednotlivá kódová slova a ukládají je do pomocných tabulek viz obr. 3.2.

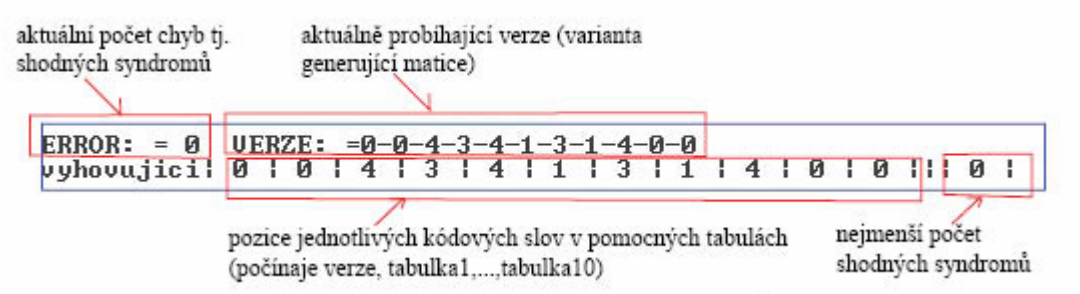
V tabulce 1 v obr. 3.2 jsou uložena všechna kódová slova s požadovanou Hammingovou vahou. Následně se vybírají ta kódová slova, která mají požadovanou min. Hammingovu váhu ke kódovému slovu 1 a ty se předávají do tabulky 2 v obr. 3.2. Jako poslední následuje údaj o počtu těchto kódových slov, aby bylo možno nastavit cyklus pro výběry slov v následující tabulce. V tabulce 2 v obr. 3.2 se opět vybírají vyhovující slova vzhledem k prvnímu kódovému slovu a opět se předávají dále. Tento děj se opakuje až do tabulky 11 v obr. 3.2, kde je uloženo jedenácté slovo. Díky tomu, že si program pamatuje pozice jednotlivých kódových slov, tak vytvoří kontrolní matici pro ověření správnosti.



Obr. 3.2: Výběr vyhovující generující matice

Další část programu testuje správnost kontrolní matice tím, že jednotlivá chybová slova násobí s kontrolní maticí a zjišťuje počet stejných syndromů.

Předposledním krokem je výpis naměřených výsledků, který udává pozice jednotlivých kódových slov v pomocných tabulkách a počet stejných syndromů. Výpis nulového počtu shodných syndromů a jednotlivé pozice vyhovujících slov je uveden na obr. 3.3.



Obr. 3.3: Výpis výsledků z programu prog_generujici_matice.cpp

Nakonec se vyhodnotí, zda jsou projita všechna kódová slova požadované Hammingovy váhy. Pokud ne, pak se program vrátí na začátek a testuje kódová slova vzhledem ke kódovému slovu 2. Při naplnění podmínky dojde k ukončení programu.

Z obr. 3.3 je patrné, že byla nalezena taková kódová slova, u kterých nedocházelo k opakování syndromů pro jednotlivá chybová slova. Po nastavení programu (jednotlivých cyklů) dle zjištěných parametrů dojde k vypsání generující matice lineárního blokového kódu schopného opravit $t = 3$ nezávislé chyby při informační

rychlosti $R = 0{,}5$. Generující matice má tedy tvar

$$[G] = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}.$$

3.2 Kontrolní matice

V předchozí kapitole byla vytvořena generující matice obecného lineárního blokového kódu schopného opravit $t = 3$ nezávislé chyby při informační rychlosti $R = 0{,}5$. Za pomoci této matice je možno vytvořit kontrolní matici potřebnou pro dekódování zabezpečeného toku dat. Při konstrukci kontrolní matice se vychází z poznatků uvedených v kapitole 2.3. Výsledný tvar kontrolní matice je tedy

$$[H] = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

Pro samotné dekódování se používá transponovaný tvar kontrolní matice. Transponovaná kontrolní matice se zapíše jako

$$\left[H^T\right]=\begin{bmatrix}
0&0&0&0&0&1&1&1&1&1&1\\
0&0&1&1&1&0&0&0&1&1&1\\
0&1&0&1&1&0&1&1&0&0&1\\
0&1&1&0&1&1&0&1&0&1&0\\
0&1&1&1&0&1&1&0&1&0&0\\
1&0&0&1&1&1&0&1&0&1&1\\
1&0&1&0&1&1&1&0&0&0&1\\
1&0&1&1&0&0&1&1&1&0&1\\
1&1&0&0&1&1&1&1&1&0&0\\
1&1&0&1&0&0&0&1&1&1&0\\
1&1&1&0&0&1&0&0&1&1&1\\
1&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}.$$

3.3 Korekce chyb

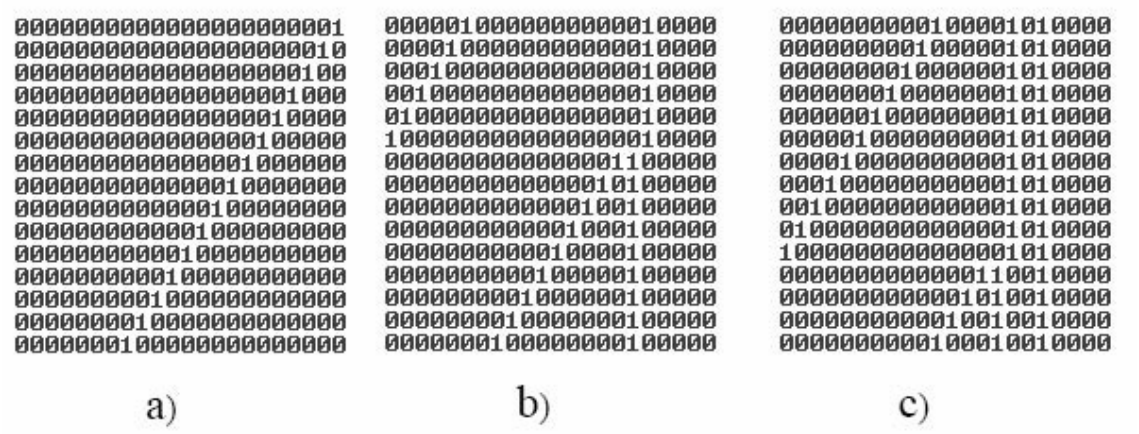
Navržený obecný lineární blokový kód je schopen opravit $t=3$ nezávislé chyby. Tato kapitola se zabývá konkrétním způsobem opravy vzniklých chyb při přenosu datového toku. Při dekódování se využívá skutečnosti, že vyjde-li při součinu vektoru přijatého slova a transponované kontrolní matice syndrom nenulový, pak došlo k chybě. Na základě syndromu se určí chybové slovo, které je následně přičteno k přijatému slovu a tím dojde k opravě chyby vzniklé při přenosu.

Zde navržený kód má celkovou délku $n=22$ a je schopen opravit $t=3$ nezávislé chyby. Za pomoci kombinatoriky lze vypočítat počet všech možných způsobů vzniku jedné, dvou a tří chyb. Výpočet celkového počtu chybových slov je

$$poč.\,chybových\,slov=\binom{n}{1}+\binom{n}{2}+\binom{n}{3}=\binom{22}{1}+\binom{22}{2}+\binom{22}{3}=$$

$$=\frac{22}{1!}+\frac{22\cdot 21}{2!}+\frac{22\cdot 21\cdot 20}{3!}=1793.$$

Počet syndromů je shodný s počtem chybových slov tj. obě tabulky obsahují 1793 řádků. Příklad některých vybraných chybových slov je znázorněn na obr. 3.4.



Obr. 3.4: Některé možné chyby a) jedna chyba, b) dvě chyby, c) tři chyby

Jsou-li známa chybová slova, pak je potřeba s jejich pomocí zjistit všechny syndromy, aby bylo s jejich pomocí možno vzniklé chyby opravovat. Za tímto účelem je vytvořen program prog_dekoder.cpp napsaný v jazyce C++.

Na začátku programu jsou určený chyby, které mohou při přenosu ovlivnit přenášenou informaci. Tato slova jsou uložena v matici `matE` a k těmto slovům v další části vypočítán příslušný syndrom. Syndromy jsou uloženy v matici `matS`. Jednotlivé řádky matice `matE`, respektive slova na daných řádcích odpovídají syndromům v příslušných řádcích matice `matS`. Program dále umožňuje opravu chybně přijatého slova, jehož správný tvar je vypsán na obrazovku. Vývojový diagram je uveden v kapitole 4, kde je navržen převodník [S]→[E], který je koncipován na základě tohoto programu.

3.4 Kódování a dekódování

Tato kapitola se zabývá konkrétním způsobem zabezpečení informace, kódováním a dekódováním obecným lineárním kódem vytvořeného v předchozí kapitole. Dále tato kapitola slouží k ověření správnosti navrženého kódu pro zadaný počet nezávislých chyb.

Přijme-li se na vstupu posloupnost $[P]=[1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1]$, pak se k jejímu zakódování využije vztah 2.11.

$$[F]=[P]\times\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&1&1&1\\
0&0&1&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&1&1&0&1&1&0&1&0&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&1&0&1&1&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&1&1&1&0&1&0&1&1\\
0&0&0&0&0&0&1&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&1&0&1&1&0&0&1&1&1&0&1\\
0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&1&1&1&1&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&0&0&0&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&1&0&0&1&1&1
\end{bmatrix}$$

$$[F]=[1\quad 0\quad 1\quad 0\quad 1\quad 0\quad 1\quad 0\quad 1\quad 0\quad 1\quad 1\quad 0\quad 1\quad 0\quad 1\quad 1\quad 1\quad 1\quad 0\quad 0\quad 0].$$

Výsledné zabezpečené slovo je tedy 10101010110101111000. Dojde-li k bezchybnému přenosu, pak je syndrom nulový a přijaté slovo $[J]$ odpovídá zabezpečenému slovu $[F]$. V případě, že při přenosu došlo k chybě, je syndrom nenulový a musí se s jeho pomocí určit chybové slovo $[E]$. Výpočet syndromu, následné určení chybového slova a oprava přijatého slova je provedena v programu prog_dekoder.cpp.

$$[S]=[1\quad 0\quad 0\quad 0\quad 1\quad 0\quad 1\quad 0\quad 1\quad 0\quad 0\quad 1\quad 0\quad 0\quad 0\quad 1\quad 1\quad 1\quad 1\quad 0\quad 0\quad 0]\times\begin{bmatrix}
0&0&0&0&0&1&1&1&1&1&1\\
0&0&1&1&1&0&0&0&1&1&1\\
0&1&0&1&1&0&1&1&0&0&1\\
0&1&1&0&1&1&0&1&0&1&0\\
0&1&1&1&0&1&1&0&1&0&0\\
1&0&0&1&1&1&0&1&0&1&1\\
1&0&1&0&1&1&1&0&0&0&1\\
1&0&1&1&0&0&1&1&1&0&1\\
1&1&0&0&1&1&1&1&1&0&0\\
1&1&0&1&0&0&0&1&1&1&0\\
1&1&1&0&0&1&0&0&1&1&1\\
1&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}$$

$$[S]=[1\quad 0\quad 0\quad 1\quad 1\quad 1\quad 1\quad 1\quad 1\quad 1\quad 0].$$

Z tabulky syndromů se zjistí, že danému syndromu odpovídá chybové slovo $[E] = [0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]$. Toto chybové slovo je přičteno ke slovu přijatému a tím se získá opravené slovo $[R] = [1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0]$. Jelikož prvních 11 bitů je informačních, tak za předpokladu max. tří chyb bylo kódovým slovem $[P] = [1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1]$.

Další ověření funkčnosti pro jiné druhy chyb je uvedeno v tab. 3.1, přičemž pátý řádek odpovídá předchozím výpočtům.

Tab. 3.1: Tabulka korekce chybně přijatých slov

vstupní slovo [F]	syndrom [S]	chybové slovo [E]	opravené slovo [R]
00101010101101011111000	00000111111	10000000000000000000000	10101010101101011111000
10101000001101011111000	01100010011	00000001010000000000000	10101010101101011111000
11101011111101011111000	01011010100	01000001010000000000000	10101010101101011111000
11101010101101011111101	00111000010	01000000000000000000101	10101010101101011111000
10001010100100011111000	10011111110	00100000001001000000000	10101010101101011111000

4 REALIZACE KODEKU

Při zabezpečování zprávy a kontrole správnosti se využívá součinu vektoru a matice. Bylo zjištěno, že tento proces je realizovatelný i pomocí zabezpečovacích součtů u kódování a kontrolních součtů u kontroly správnosti přenosu. Při realizaci kodéru se zde vychází z již vytvořeného obecného lineárního blokového kódu vytvořeného v kapitole 3.1 a 3.2.

4.1 Kodér

Kodér se využívá k zakódování zprávy. V tomto případě zabezpečuje přenášenou zprávu vůči výskytu maximálně tří chyb. Zapojení kodéru vychází z generující matice [4]. Konkrétně se jedná o způsob vytvoření zabezpečovacích prvků. Ty se vytváří pomocí součtů mod 2 jednotlivých řádků zabezpečovací podmatice, obsahující jedničky. Jednotlivé zabezpečovací prvky jsou pak dány

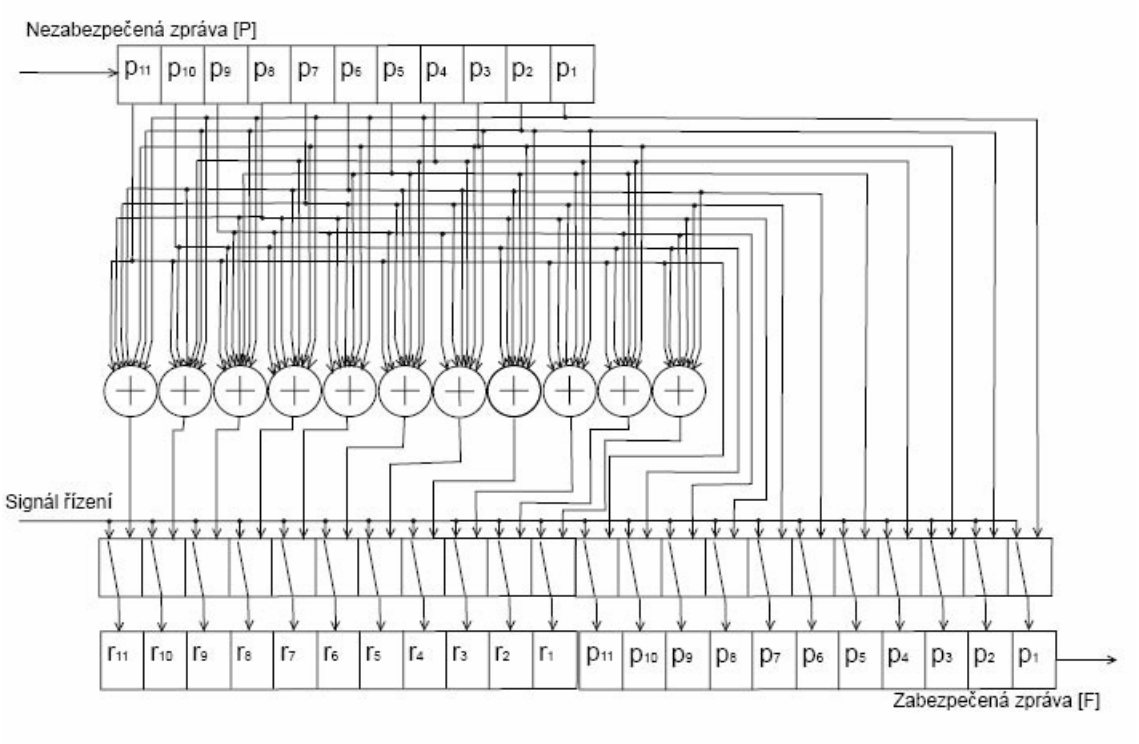
$$[G]=\left[\begin{array}{cccccccccccc:cccccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&1&1&1\\
0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&1&1&0&1&0&1&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1&0&1&1&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&1&1&1&0&1&0&1&1\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&1&1&0&0&1&1&1&0&1\\
0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&1&1&1&1&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&1&1&0&0&1&0&0&1&1&1
\end{array}\right]\begin{matrix}p_1\\p_2\\p_3\\p_4\\p_5\\p_6\\p_7\\p_8\\p_9\\p_{10}\\p_{11}\end{matrix}$$

Výsledné součty mod 2 jednotlivých řádků zabezpečovací podmatice (pravá polovina) udávající zabezpečovací prvky jsou uvedeny v tab. 4.1.

Tab. 4.1: Tabulka zabezpečovacích prvků

Zabezpeč. prvek	součet mod 2	Zabezpeč. prvek	součet mod 2
r_1	$p_6+p_7+p_8+p_9+p_{10}+p_{11}$	r_7	$p_1+p_3+p_5+p_7+p_8+p_9$
r_2	$p_3+p_4+p_5+p_9+p_{10}+p_{11}$	r_8	$p_1+p_3+p_4+p_6+p_8+p_9+p_{10}$
r_3	$p_2+p_4+p_5+p_7+p_8+p_{11}$	r_9	$p_1+p_2+p_5+p_8+p_9+p_{10}+p_{11}$
r_4	$p_2+p_3+p_5+p_6+p_8+p_{10}$	r_{10}	$p_1+p_2+p_4+p_6+p_{10}+p_{11}$
r_5	$p_2+p_3+p_4+p_6+p_7+p_9$	r_{11}	$p_1+p_2+p_3+p_6+p_7+p_8+p_{11}$
r_6	$p_1+p_4+p_5+p_6+p_7+p_9+p_{11}$		

Jsou-li známy zabezpečovací prvky, pak lze odvodit zapojení kodéru, které je znázorněno na obr. 4.1.



Obr. 4.1: Zapojení kodéru obecného lineárního blokového kódu (22;11)

4.2 Dekodér

Základní funkcí dekodéru je dekódovat příchozí zprávu. V tomto případě se jedná o kontrolu správnosti přenosu, popřípadě nastane-li chyba, jedná-li se tedy o maximálně tři chyby, je úkolem dekodéru tyto chyby opravit a na výstup poslat opravené slovo.

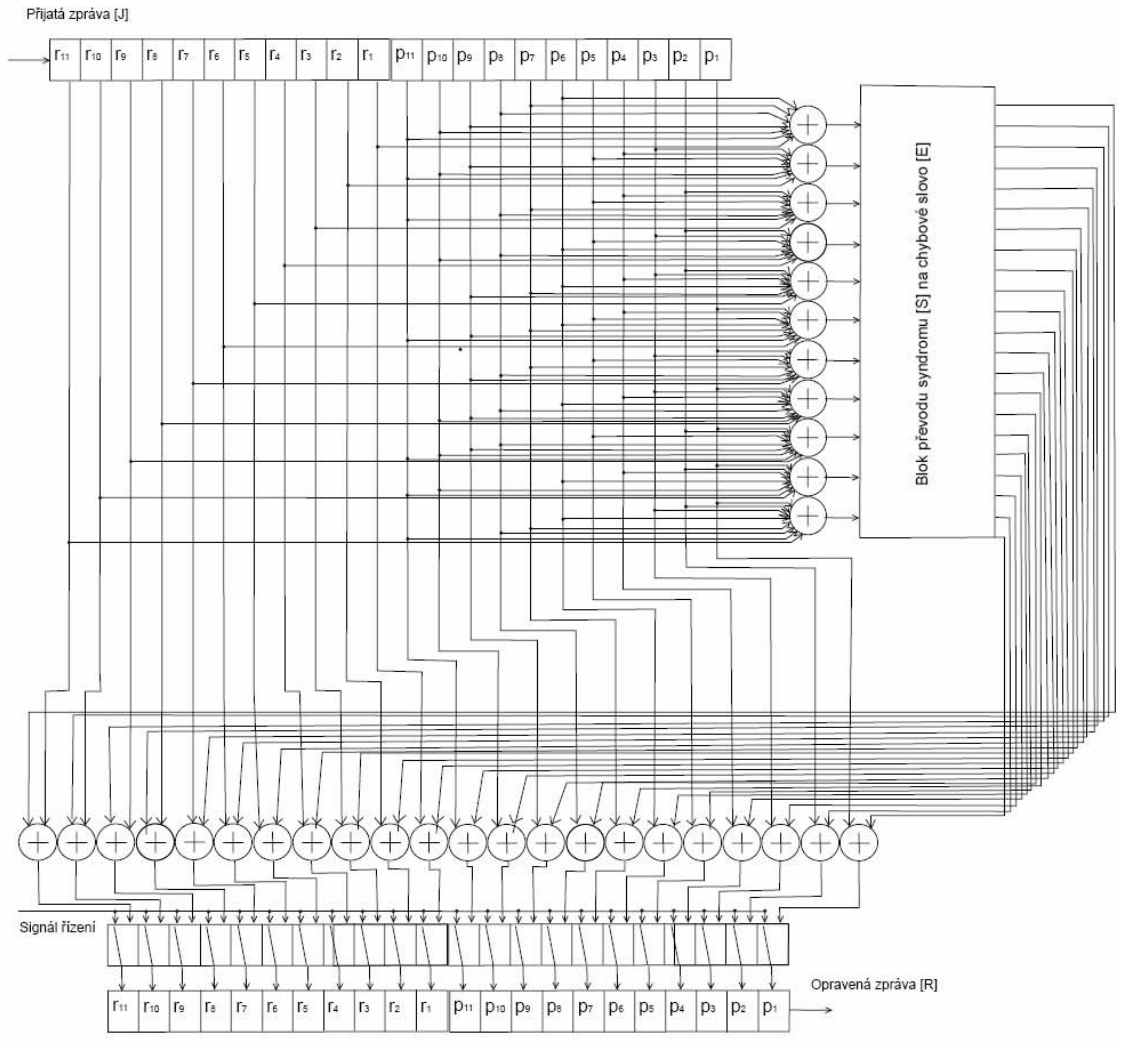
Způsob zapojení dekodéru vychází z kontrolní matice [4]. Zde se vychází ze způsobu určení prvků syndromů. Prvek syndrom je dán součtem mod 2 prvků přenesené kódové kombinace, které mají v příslušném řádku kontrolní matice hodnotu jedna. Odvozené prvky syndromu jsou uvedeny níže, opět se zde vychází z kontrolní matice vytvořené v předchozí kapitole.

Tab. 4.2: Tabulka prvků syndromů

prvek syndromu	součet mod 2	prvek syndromu	součet mod 2
s_1	$p_6+p_7+p_8+p_9+p_{10}+p_{11}+r_1$	s_7	$p_1+p_3+p_5+p_7+p_8+p_9+r_8$
s_2	$p_3+p_4+p_5+p_9+p_{10}+p_{11}+r_2$	s_8	$p_1+p_3+p_4+p_6+p_8+p_9+p_{10}+r_8$
s_3	$p_2+p_4+p_5+p_7+p_8+p_{11}+r_3$	s_9	$p_1+p_2+p_5+p_8+p_9+p_{10}+p_{11}+r_9$
s_4	$p_2+p_3+p_5+p_6+p_8+p_{10}+r_4$	s_{10}	$p_1+p_2+p_4+p_6+p_{10}+p_{11}+r_{10}$
s_5	$p_2+p_3+p_4+p_6+p_7+p_9+p_5$	s_{11}	$p_1+p_2+p_3+p_6+p_7+p_8+p_{11}+r_{11}$
s_6	$p_1+p_4+p_5+p_6+p_7+p_9+p_{11}+r_6$		

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

Pomocí součtů z tab. 4.2 se sestaví obvod pro určení syndromu. Za pomoci syndromu se následně odvodí chybové slovo, které součtem mod 2 s přijatým slovem dává opravenou kombinaci. Příklad odvozeného zapojení je uvedeno na obr. 4.2.



Obr. 4.2: Zapojení dekodéru obecného lineárního blokového kódu (22;11)

Z obr. 4.2 je patrné, že oblast, přiřazující chybové slovo k syndromu je realizována blokem „blok převodu syndromu [S] na chybové slovo [E]“, na jehož vstup je přivedeno 11 hodnot reprezentujících syndrom a na jeho výstupu je zjištěné chybové slovo jehož délka je 22 bitů. Z důvodu složitosti tohoto bloku je mu věnována další kapitola.

4.3 Převodník [S]→[E]

Převodník [S] → [E] byl již zmíněn v předchozí kapitole. Jedná se o blok obsažený v dekodéru, jehož úkolem je převést syndrom na chybové slovo, které umožní opravit chybně přijaté slovo. Předpokladem je výskyt maximálně tří nezávislých chyb.

U jednoduchých dekodéru tj. dekodér schopný opravit jen jednu chybu stačí tento blok realizovat pomocí obvodu logického součinu. Při větším množství chyb je tato varianta nevyhovující a proto je potřeba k tomuto účelu využít jiný postup.

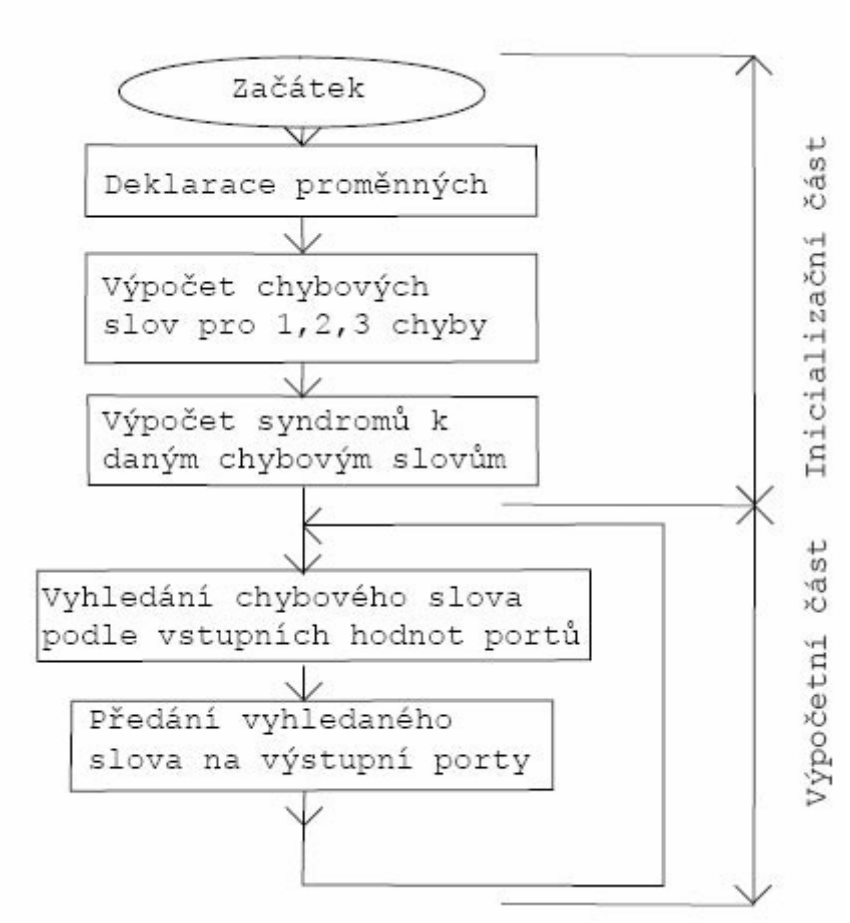
Při návrhu převodníku byl využit postup využívající mikrokontroler. Současné mikrokontrolery lze programovat v jazyce assembler, C a C++. V této práci byl zvolen programovací jazyk C z důvodu

1) Již hotový program – v kapitole 3.3 byl vytvořen program prog_dekoder.cpp, který je již napsán v jazyce C++ a tudíž není nutné jej přepisovat do assembleru (za pomoci malých úprav je možné tento program použit v mikrokontroleru).
2) Přehlednost – programy psané v jazyce C jsou přehlednější a vystihující pravou podstatu programu.
3) Přenositelnost – programy jsou přenositelné mezi různými typy mikrokontrolerů.

Vytvořený program, který je použit v mikrokontroleru se nazývá prog_mikrokontroler. Program a tím i činnost mikrokontroleru lze rozdělit do dvou částí a to inicializační a výpočetní viz vývojový diagram na obr. 4.3.

Úkolem inicializační části je deklarovat všechny proměnné, ale hlavně vypočítat kombinaci všech možných chyb pro maximálně tři současně se vyskytující. Tyto chyby následně zapisuje do tabulky `matE` a dále výpočet příslušných syndromů, které jsou ukládány do `matS`. Celý cyklus se provede pouze jednou a to při spuštění mikrokotroléru (přivedením napájecího napětí).

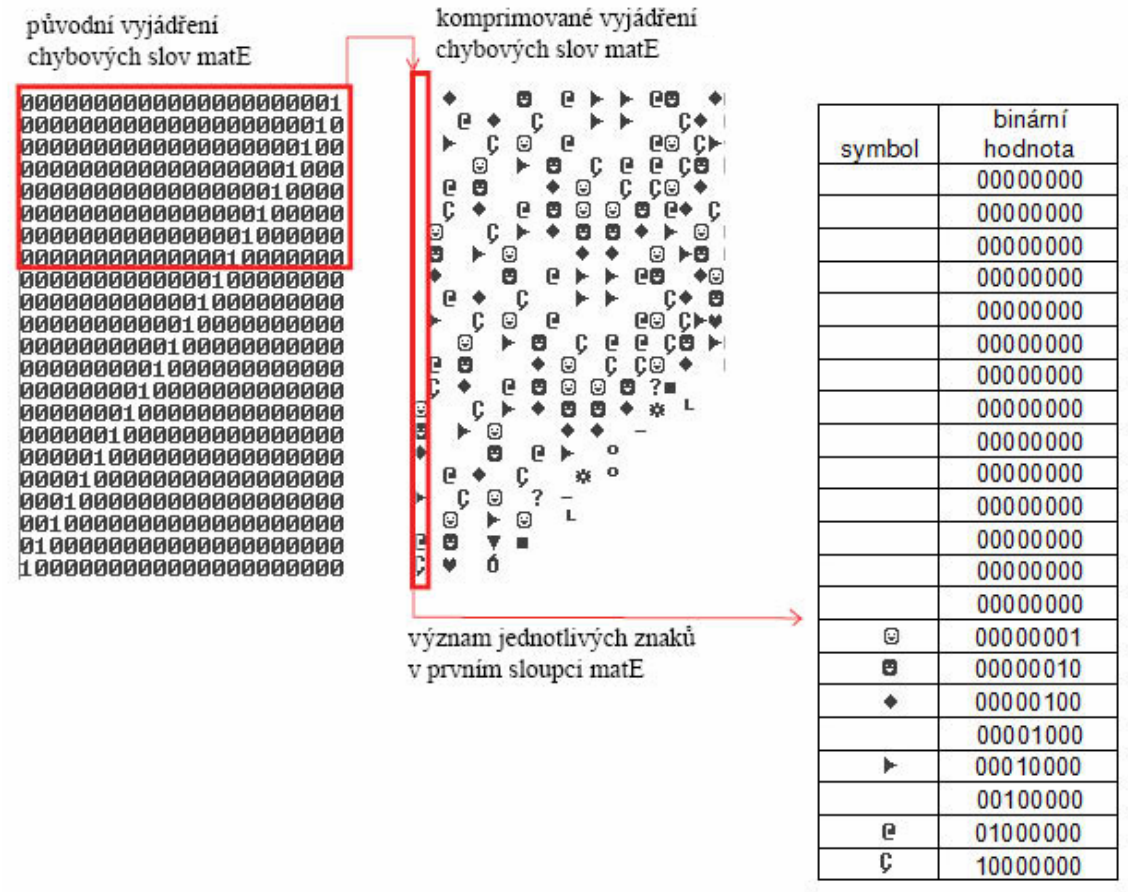
Další výpočetní část se stará pouze o to, že zjišťuje hodnoty na vstupních portech [9] (v tomto případě se jedná o porty PORTA(PINA7 až PINA0) a PORTB(PINB7 až PINB5)), tyto hodnoty udávají hodnotu syndromu. Zjištěný syndrom je pak vyhledáván v matici `matS` a po shodě je na výstup posláno chybové slovo odpovídající stejnému řádku v matici `matE`. Výstup je zde realizován třemi porty, konkrétně se jedná o port PORTC (PINC7 až PINC0), PORTD (PIND7 až PIND0) a PORTE (PINE7 až PINE2). Hodnoty těchto portů udávají chybové slovo, které je výstupem z mikrokontroleru a následně slouží pro korekci vzniklých chyb v přijatém slově. Uvedený cyklus se cyklicky opakuje.



Obr. 4.3: Vývojový diagram programu převodníku [S]→[E]

Jak již bylo zmíněno, jádro programu použitého v mikrokontroleru vychází z programu prog_dekoder.cpp využitého v kapitole 3.3. Aby bylo možno tento program využít i pro praktické účely, je potřeba provést následující změny

1) Změna deklarace – jedná se o první a základní změnu oproti předchozímu programu. Jelikož jsou součástí programu tabulky (matice), které obsahují chybová slova a syndromy, které mají počet hodnot (řádků) 1793 a jsou deklarovány jako `int`, vychází-li se z toho, že velikost int pro většinu 8 bitových kompilátorů je 16 bitů, pak jen jejich velikost činí cca 115,56 kB ($\frac{16 \cdot 1793 \cdot 22}{8 \cdot 1024} kB + \frac{16 \cdot 1793 \cdot 11}{8 \cdot 1024} kB$). Takováto velikost je pro mikrokontroler nepředstavitelná a proto je potřeba ji redukovat. Docílilo se toho použitím deklarace `char`, jenž má velikost 8 bitů, a tím vznikne matice charů, která má 225 řádků a celková velikost obou matic vychází na cca 7,22 kB.
2) Přístup k prvkům – z důvodu změny struktury matic chybových slov a syndromů, je potřeba změnit i přístup k jednotlivým hodnotám, jelikož jedna buňka matice neobsahuje bit daného slova, ale osmici bitů slov sousedních na dané pozici viz obr. 4.4.



Obr. 4.4: Komprimace matice vhodná pro mikrokontroler

Z obrázků je patrné, že k jednotlivým chybovým slovům se musí přistupovat po bitech. Tento přístup je zabezpečen pomocí logického součtu, kdy se zapisuje na požadovaný bit pomocí přičtení k danému znaku binární hodnotu dané pozice tj. na první bit je přičteno 1d, na druhý 2d, na třetí 4d až osmý 128d. Při čtení potřebných bitů je postup stejný, rozdíl je jen v použití logického součinu oproti logickému součtu.

3) Zrušení výpisů – je potřeba zrušit výpisy na obrazovku, jelikož nejsou podporovány mikrokontrolerem.

4) Nastavení I/O – je potřeba nastavit vstupně výstupní porty. Jako vstupní porty jsou nastaveny PORTA a PORTB, kde nastavení výstupního charakteru je dáno DDRA=0 a DDRB=0. Výstupními porty jsou PORTC, PORTD a PORTE nastaveny pomocí DDRC=255, DDRD=255 a DDRE=255.

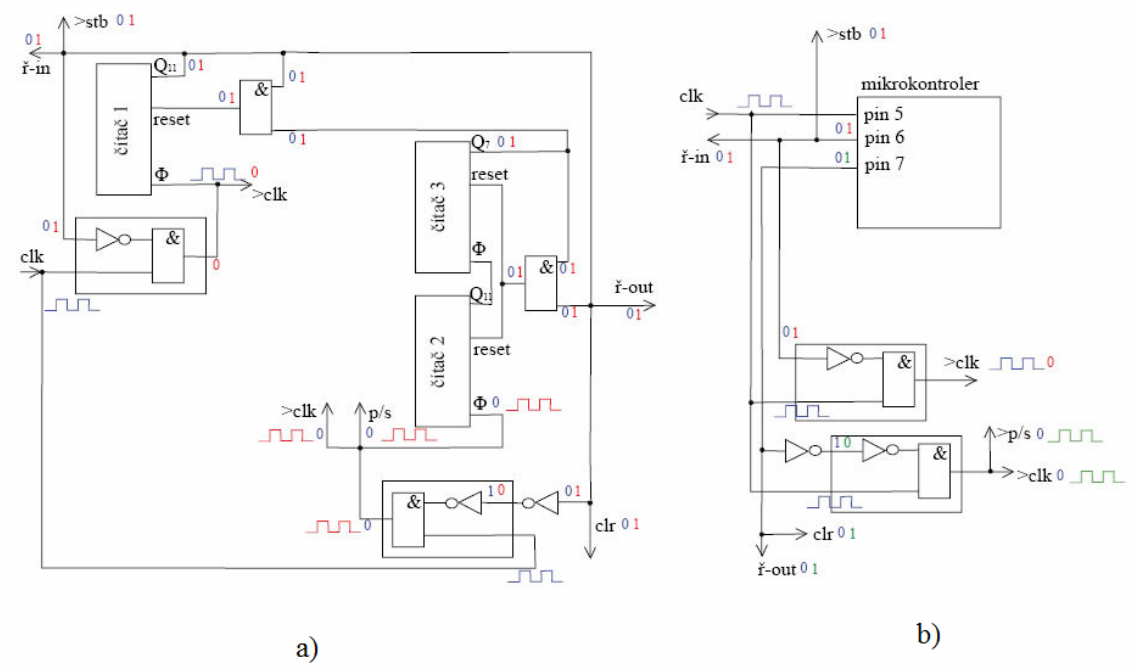
Výsledný modifikovaný program je použit v mikrokontroléru, který splňuje požadované vlastnosti pro výskyt maximálně tří nezávislých chyb. Pokud se vyskytne jejich syndrom, pak je bezchybně určeno chybové slovo.

4.4 Řídící obvod

Jedná se o obvod, provádějící komunikaci mezi zde navrženým kodekem a vnějším systémem. Řídící obvod dává systému najevo, že může posílat data přivedením log 0 na vodič Ř-in a naopak přivedením log 1 pro zákaz posílání dat. Obdobný způsob je na výstupu obvodu. Systému sdělí obvod log 0 nečinnost a log 1 vysílání informací.

Zapojení řídícího obvodu kodéru je na obr. 4.5 a). Na obrázku jsou dále vidět jednotlivé průběhy, kde modře je vyjádřen počáteční stav a červeně změna stavu.

Čítač má v obvodu za úkol napočítat 11 bitů podle synchronizovaného clk (clock) signálu. Po tuto dobu je systému povoleno posílat data tzn. signalizace log 0. Při napočítání jedenácti impulzů, dochází ke změně stavu na log 1, systém pozastaví posílání dat a data, která jsou uložena v pamětech s/p převodníku kodéru jsou převedena na výstup pomocí náběžné hrany >stb. Dále je spuštěn čítač do 22 a předáno hlášení o vysílání systému. Signály >clk a p/s slouží k zápisu dat do posuvných registrů a na výstup p/s převodníku. Po napočítání hodnoty 22, je na výstupu Q_7 čítače 3 hodnota log 1, která má za následek resetování celého obvodu a tudíž návrat do výchozího stavu.



Obr. 4.5: Zapojení a průběhy řídícího obvodu a) kodéru, b) dekodéru

Struktura řídícího obvodu dekodéru je jednodušší, jelikož odpadají čítače, což je dáno výskytem mikrokontroleru v obvodu dekodéru. Zapojení řídícího obvodu je na obr. 4.5 b) a opět jsou zde uvedeny hodnoty v jednotlivých bodech. Aby obvod správně fungoval, je zapotřebí, aby zdrojový kód vněm vložený obsahoval příkazy níže.

```
DDRK=222;
PORTK=255;
```

Pomocí prvního příkazu se nastaví pin 5 jako vstupní a tudíž je na něj možno posílat signál clk, pin 6 a pin 7 jsou nastaveny jako výstupní. Druhý příkaz pošle na piny 6 a 7 hodnotu log 1. Tyto dva příkazy jsou umístěny na začátku celého programu.

Dále je potřeba zadat níže uvedenou zdrojovou část.

```
PORTK=0;
for(i=0;i<11;i++) {
     x=1;
     y=1;
     while (x==1 || y==1) {
     pozicebit=PORTK&32;
     if(pozicebit==32)
       x=0;
     else y=0;
      }
 }
 PORTK+=64;
 pozicebit=0;
```

Tato část programu nahrazuje první čítač a je umístěna v programu za inicializační částí tj. po skončení výpočtů syndromů.

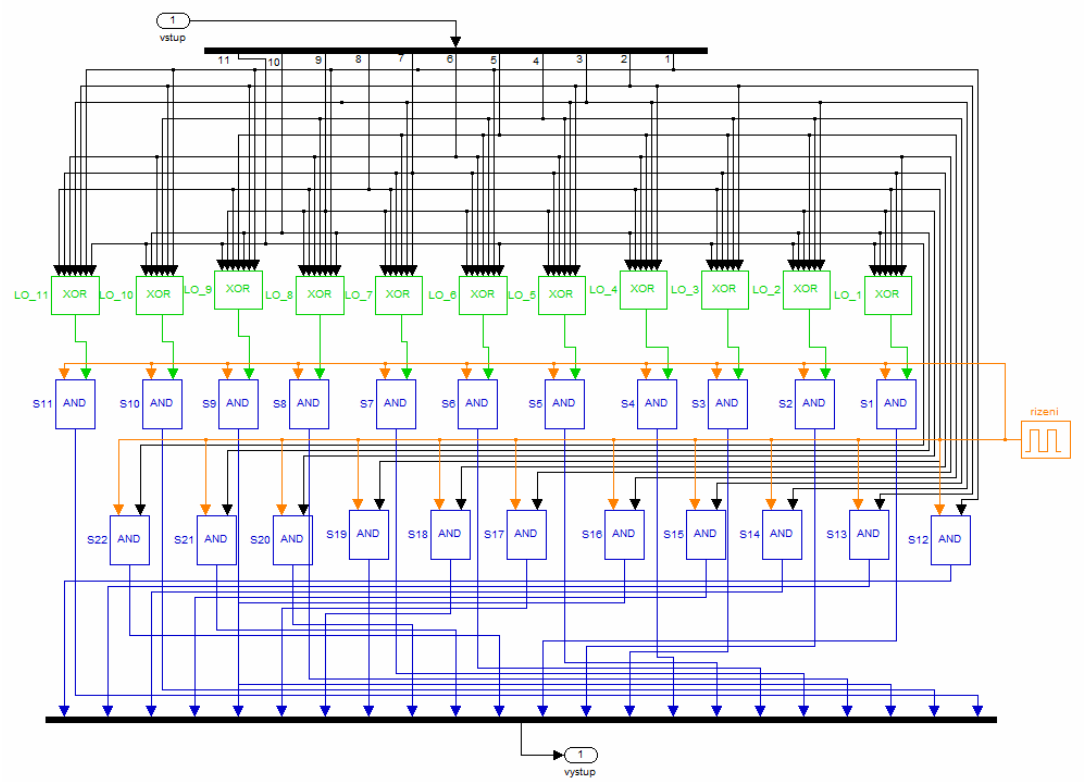
Poslední částí je nahrazující čítače na výstupů dekodéru, program má stejnou strukturu jako předchozí zdrojový kód, jen s tím rozdílem, že se zde čítá do hodnoty 22 a signalizační log hodnoty se posílá na pin 6.

```
PORTK+=128;
for(i=0;i<22;i++) {
     x=1;
     y=1;
     while (x==1 || y==1) {
     pozicebit=PORTK&32;
     if(pozicebit==32)
       x=0;
     else y=0;
     }
 }
 PORTK+=128;
 pozicebit=0;
```

5 OVĚŘENÍ FUNKČNOSTI

V předchozích kapitolách byly rozebrány postupy vytvoření obecného blokového kódu a vytvoření zapojení kodéru a dekodéru. Tato kapitola se zabývá dalším úkolem obsaženém v zadání a to ověření funkční schopnosti tohoto kodeku. Pro simulaci byl zvolen program Matlab Simulink v7.5.0.

Zapojení kodéru v programu Matlab je uvedeno na obr. 5.1. Princip zapojení tohoto obvodu je odvozen z obr. 4.1. Na vstup kodéru je přivedena nezabezpečená zpráva, jejíž délka činí 11 bitů ($[P]=[10101010101]$), tyto bity jsou zadány jako konstanta a jsou pomocí bloku demux rozděleny do jedenácti větví. Pomocí logických obvodů XOR realizujících součet mod 2 a obvodů AND sloužící jako spínače dochází k zabezpečení zprávy a jejímu předání přes blok mux na výstup, kde je vyslána zabezpečená posloupnost $[P]=[10101010110110111000]$ na přenosový kanál.

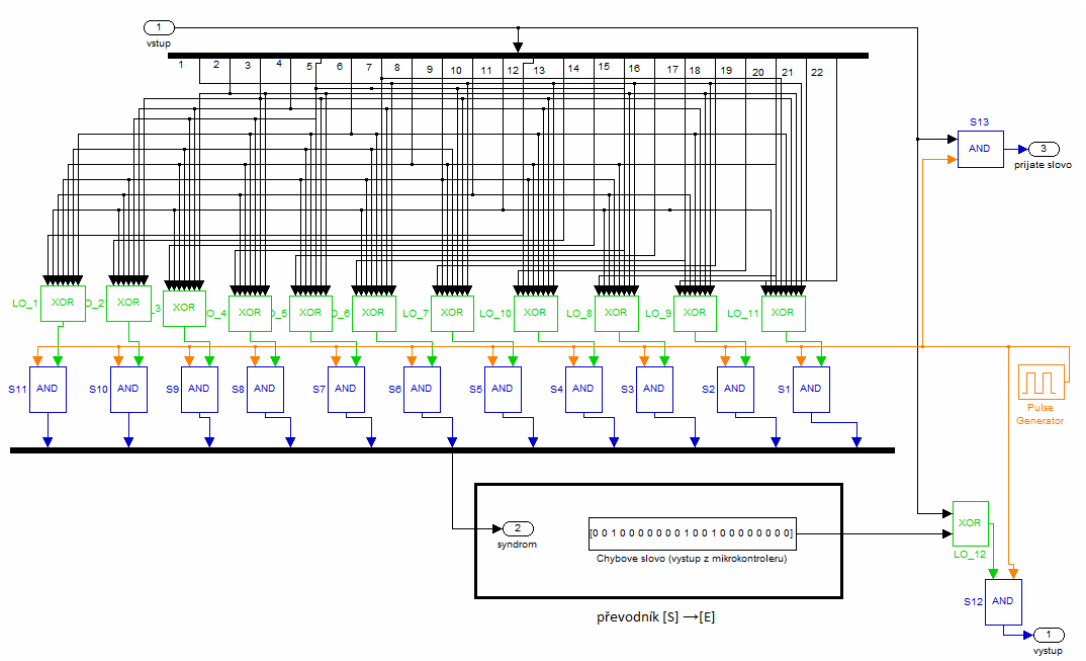


Obr. 5.1: Zapojení kodéru v programu Matlab Simulink

Mezi blokem kodéru a dekodéru je nasimulována chyba přičtením chybového slova k slovu vyslanému kodérem.

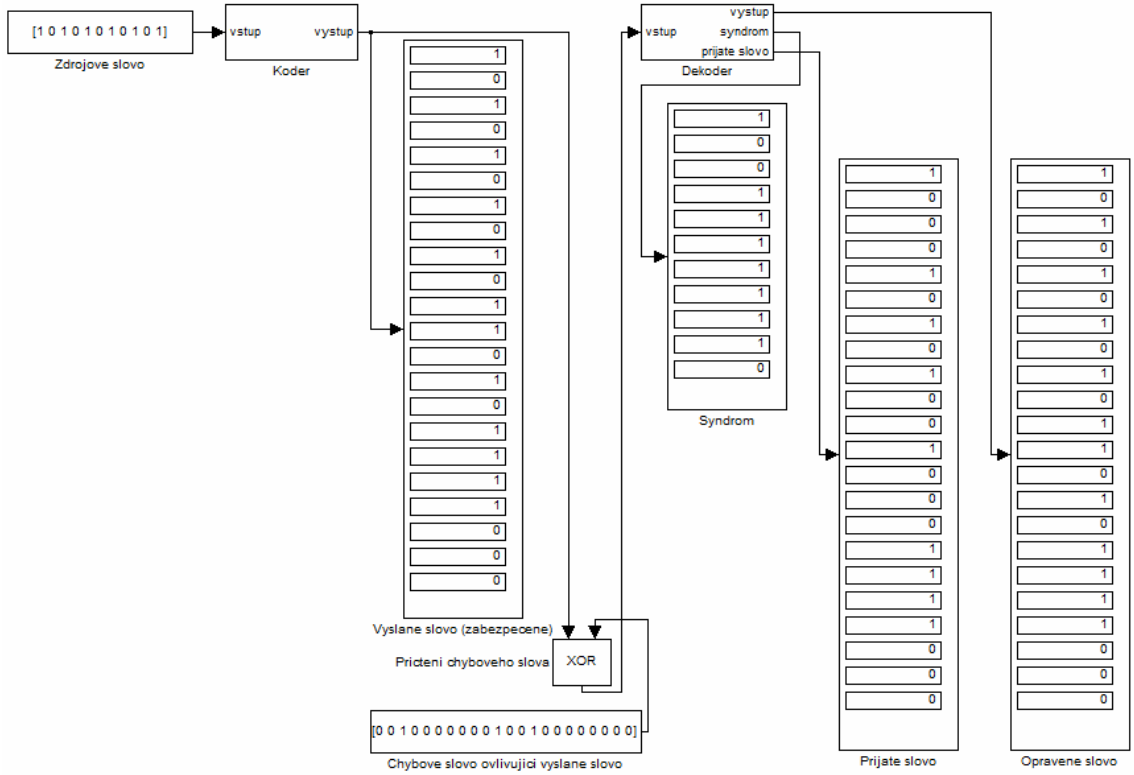
Zapojení dekodéru v Matlabu je znázorněno na obr. 5.2 a) vychází se zde z návrhu dekodéru viz obr. 4.2. Na vstup dekodéru je přivedeno přijaté slovo, na jehož základě je zjištěn syndrom. Hodnota syndromu je přivedena do převodníku [S]→[E], který je zde zobrazen pouze jako blok a musí se zde zadat chybové slovo manuálně, jelikož je tento blok simulován samostatně v jiném programu. Z bloku převodníku [S]→[E] je na

výstupu vysláno chybové slovo, které je přičtené k přijatému a tím dochází k opravě vzniklých chyb při přenosu.



Obr. 5.2: Zapojení dekodéru v programu Matlab Simulink

Uvedený průběh je z důvodu přehlednosti vyjádřen blokově na obr. 5.3, kde jsou zobrazeny i konkrétní hodnoty bitových posloupností v důležitých bodech.



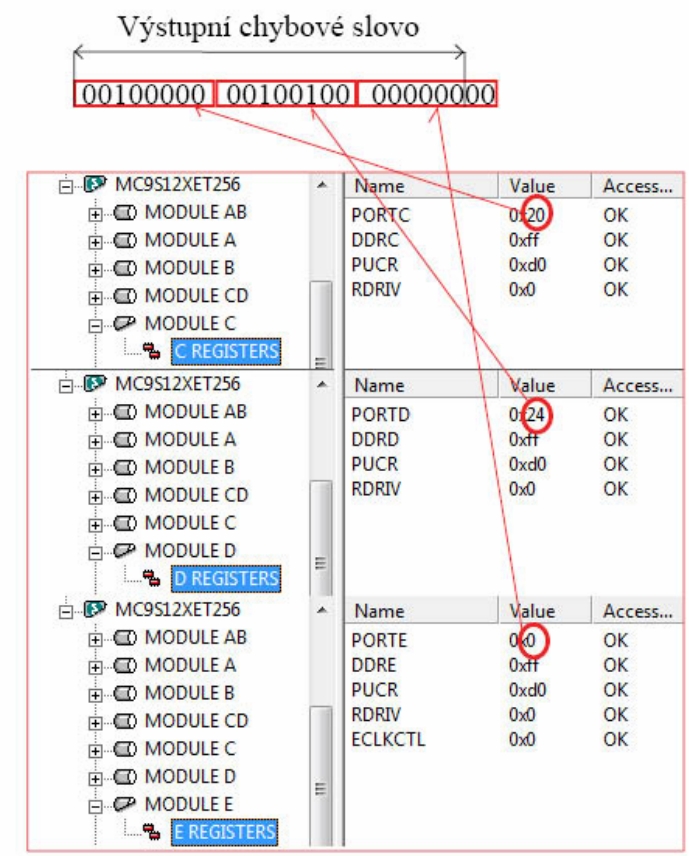
Obr. 5.3: Blokové zapojení kodeku v programu Matlab Simulink

Při ověřování funkčnosti kodeku v programu Matlab, bylo řečeno, že do bloku převodníku [S]→[E] se zadává hodnota chybového slova manuálně. Jelikož se jedná o podstatnou část obvodu dekodéru, je potřeba odsymulovat i tuto část. Za tímto účelem byl použit program pro psaní a ověřování funkčnosti kódu pro mikrokontrolery. Jedná se o program Freescale CodeWarrior development Studio for S12(X) V5.0. Tento program umožňuje simulaci Freescale mikrokontrolerů řady 12.

Po zpuštění programu byl vybrán mikrokontroler MC9S12XET256, který byl zvolen na základě velikosti paměti RAM dostačující pro zde vytvořený program prog_mikrokontroler. Zdrojový kód byl vložen do mikrokontroleru, kde byl zpuštěn. Jelikož během činnosti programu nelze zadávat vstupní parametry, tak byly nastaveny před zpuštěním (tj. PORTA a PORTB jsou nastaveny jako výstupní!!!).

Samotné ověření lze realizovat pomocí LED diod připojených na příslušné výstupní piny, které by signalizovaly logickou jedničku rozsvícením. Tento způsob simulace však nebyl možný z důvodu použití demoverze programu. Z tohoto důvodu byla použita metoda čtení registrů, které odpovídají výstupním portům PORTC, PODTD a PORTE.

K registrům je možný přístup přes Component → Open…→Inspect, kde se vybere IO Register → MC9S12XET256. Zde je již možno přistupovat k jednotlivým registrům portů. Hodnota portu je udávána v hexadecimálním tvaru, tudíž je nutno ji převést na binární tvar, jak je znázorněno na obr. 5.4.



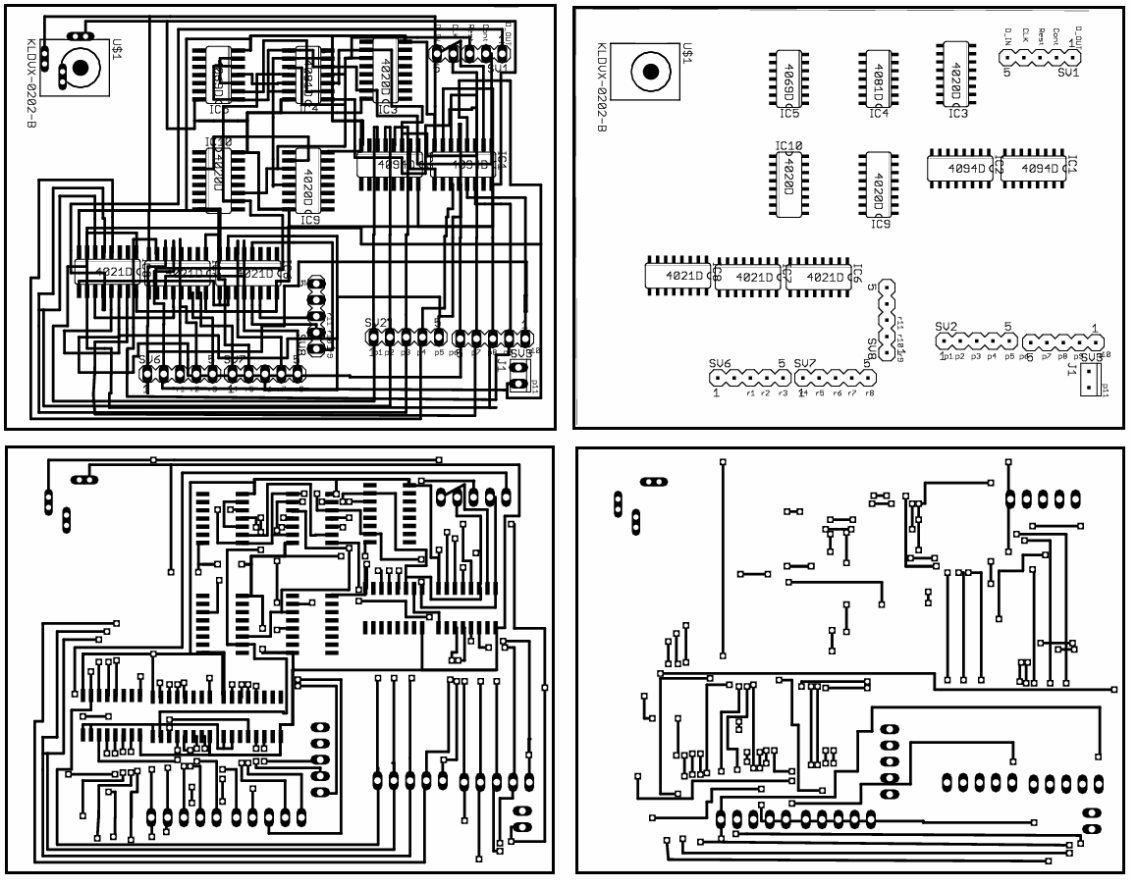
Obr. 5.4: Určení chybového slova v programu Freescale CodeWarrior

6 NÁVRH PLOŠNÉHO SPOJE

Při návrhu plošného spoje se vychází z navrženého kodeku. Z důvodu složitosti obvodu, kdy je použito velké množství integrovaných obvodů a za použití freewarového programu, který přináší značná omezení, jako je maximální velikost navrhované desky plošného spoje a maximální počet zapojených součástech na desce, bylo nutno jednotlivé části kodeku tj. kodér a také dekodér rozdělit na více desek. Při návrhu byl využit freewarový program pro tvorbu plošných spojů EAGLE 5.9.0.

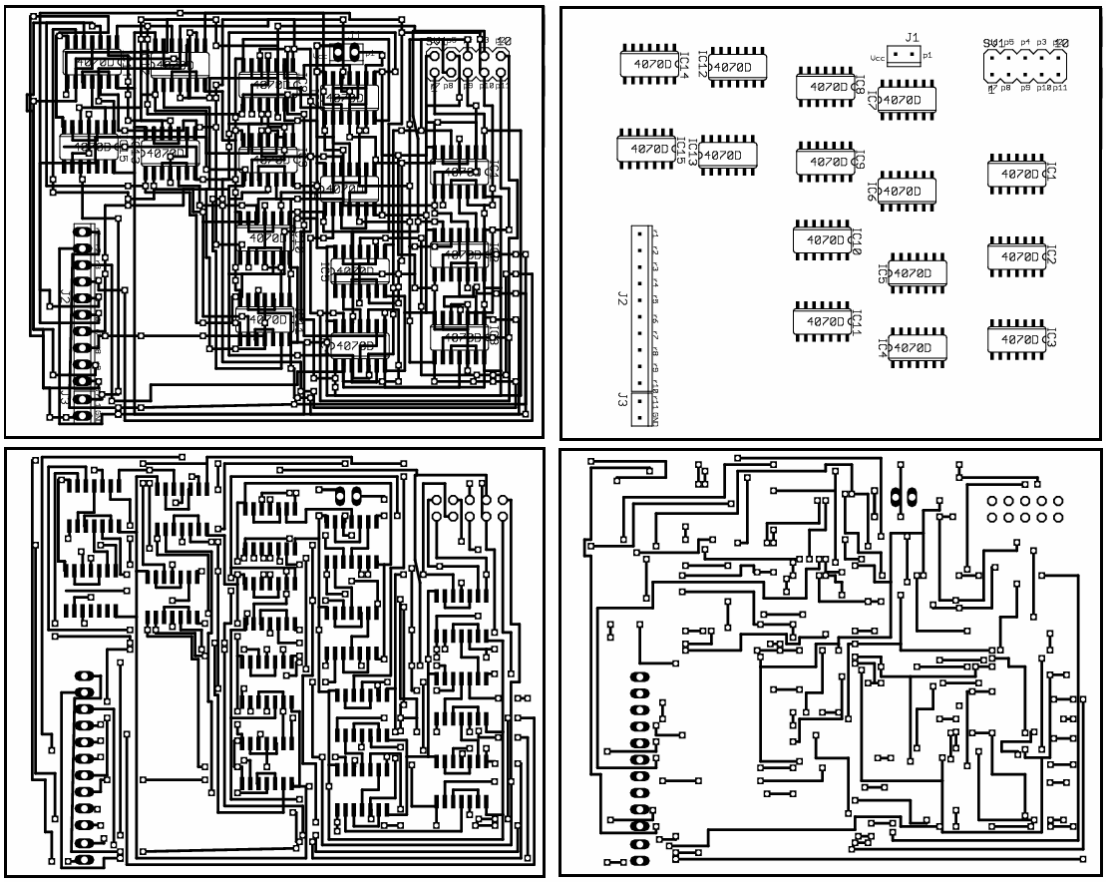
Část dekodéru reprezentující řídící obvod a obvod rozdělení sériového toku na paralelní je umístěna na desce plošného spoje na obr 6.1. Jsou zde vstupy pro sériový tok dat D_IN, synchronizační hodiny CLK a restart celého kodéru Rest. Dále jsou zde výstupy pro sériový výstup zakódovaných dat D_OUT a řídící výstup Cont. Obvod má dále vstup na napájení, které je vyžadováno 5V.

Obvodové řešení je odvozeno z obr 4.5 a), kde jednotlivé čítače jsou realizovány integrovanými obvody 4020D, na logický součin jsou použity obvody 4081D, investory reprezentuje obvod 4069D. In tygrovaný obvod 4094 reprezentuje sériově-paralelní převodník a 4021 paralelně-sériový převodník.



Obr. 6.1: Zapojení kodéru lineárního blokového kódu (22;11) – část řídící

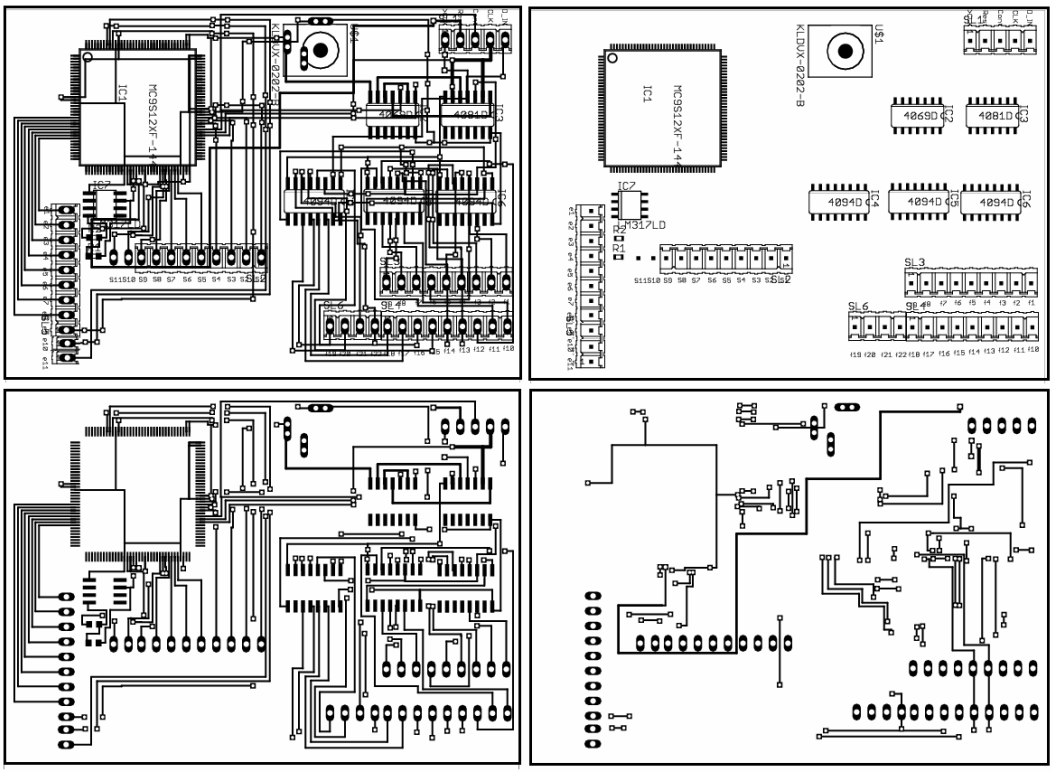
Druhá část obvodů realizující samotné kódování je uvedena na obr. 6.2. Tato část provádí operace exklusivního součtu pomocí hradel 4070D. Jedná se o realizaci obr 4.1. Tento obvod vrací řídícímu obvodu zabezpečovací prvky a ten pak k těmto prvkům přidá informační prvky a společně je pošle na výstup. Řízení přesunu zabezpečené informace na výstup, je zde zabezpečeno paralelně sériovým převodníkem.



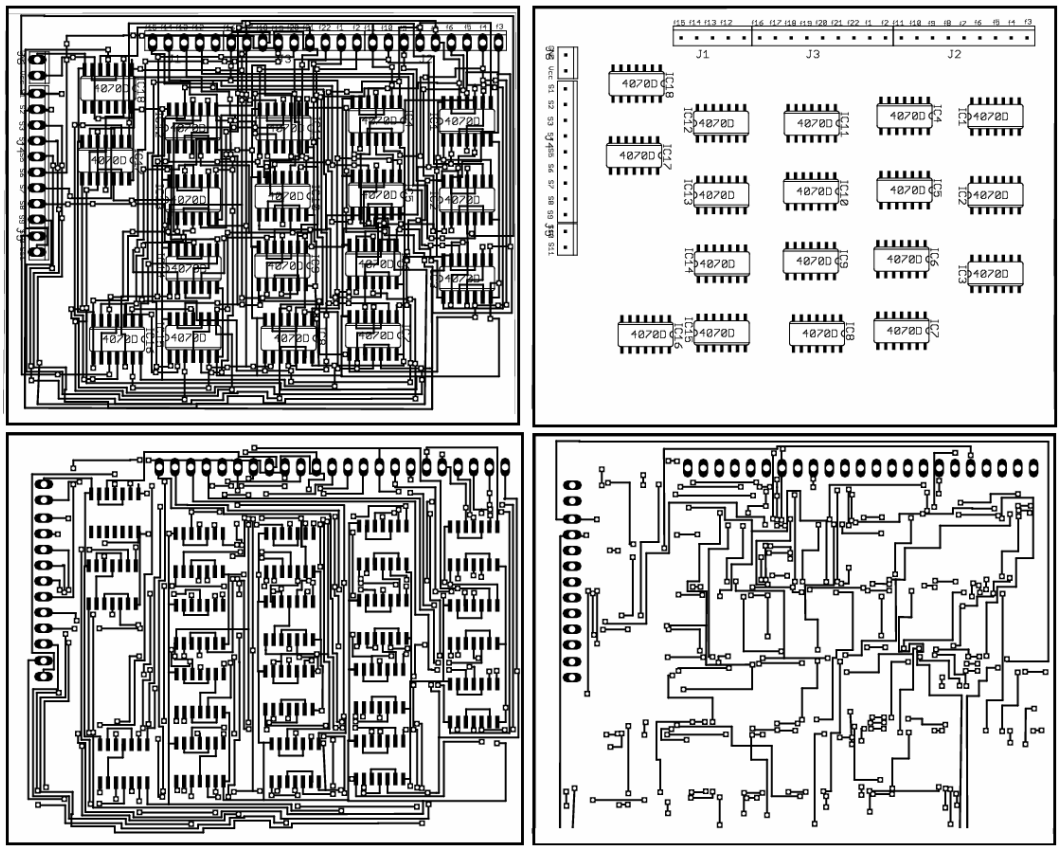
Obr. 6.2: Zapojení kodéru lineárního blokového kódu (22;11) – kódovací část

Návrh desky dekodéru je obdobný jako u kodéru. I v tomto případě je deska rozdělena na části. Na obr. 6.3 je znázorněna řídící část zabezpečující opět řízení a převody. Jsou zde obsaženy vstupy pro sériový vstup D_IN, synchronizační hodiny CLK a možný restart dekodéu Rest. Výstupními hodnotami jsou pak řízení Cont, sdělující systému posílání dat na výstupu nebo možnost příjmu dat a ještě výstup >CLK. Tento výstup souvisí s řízením a je určen dekodéru, konkrétně části s paralelně-sériovým převodníkem posílající data na výstup. Obvod vyžaduje napájení 5V a vychází ze zapojení obrázku 4.5 b).

Jako mikrokontrolér reprezentující převod syndromu na chybové slovo a řídící část obvodu, byl zvolen typ MC9S12XF-144 a to zvláště kvůli velikosti paměti RAM 16 kB, která plně postačuje zde vytvořenému programu 8 kB. Integrované obvody logického součinu a investorů jsou opět obvody 4081D a 4069D. Z důvodu potřeby dalšího napájení mikrokontroléru byl použit stabilizátor M317LD, který vytváří za pomoci rezistorů R1=240Ω a R2=110Ω napětí 1,8V. Poslední částí jsou integrované obvody 4094D realizující převod na paralelní přenos.



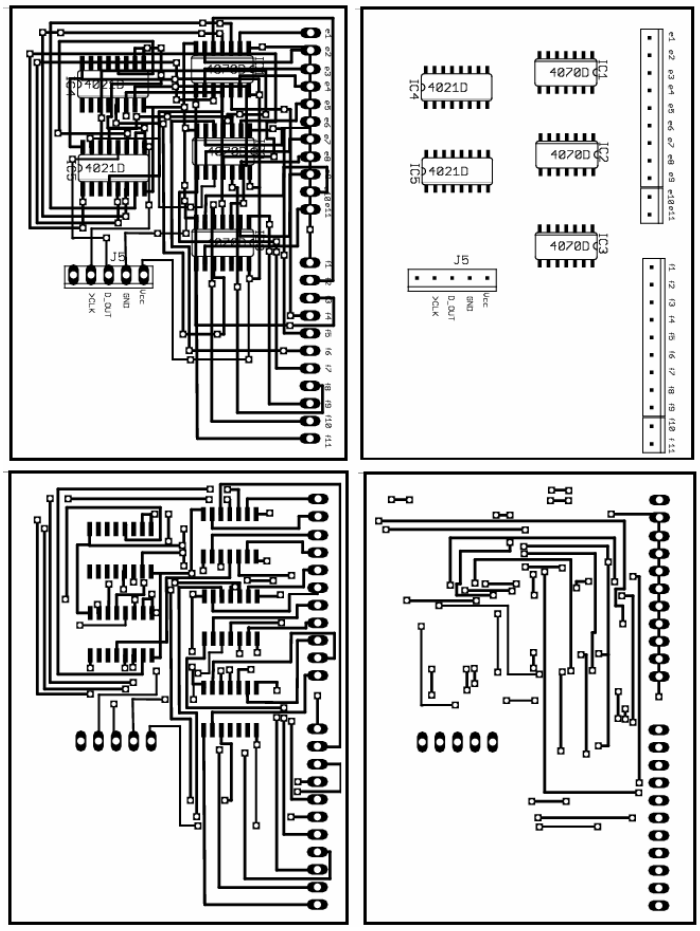
Obr. 6.3: Zapojení dekodéru lineárního blokového kódu (22;11) – část řídící



Obr. 6.4: Zapojení dekodéru lineárního blokového kódu (22;11) – dekódovací část

Obvod na obr. 6.4 je deska zabezpečující dekódování, konkrétně se jedná o výpočet syndromu, který je zpětně poslán řídící části, kde jej zpracuje mikrokontrolér. Celé zapojení je pouze realizace exklusivního součtu pomocí integrovaných obvodů 4070D.

Poslední část dekodéru je na obr. 6.5. Tento obvod slouží k případné opravě chyby. Na vstupy je přivedena první polovina chybového slova od mikrokontroléru a první polovina přijaté zprávy zastupující informační bity. Obvod na základě integrovaných exklusivních součtů 4070D provede opravu a přes obvod 4020D je dekódovaná zpráva poslána sériově na výstup.



Obr. 6.5: Zapojení dekodéru lineárního blokového kódu (22;11) – korekční část

7 ZÁVĚR

V souladu se zadáním, řeší bakalářská práce problematiku zabezpečení přenosu dat obecnými lineárními blokovými kódy. Podstata řešení problematiky je uvedena v třetí až šesté kapitole této práce.

Jedním z úkolů dle zadání bylo navrhnout obecný lineární blokový kód, který je schopný zabezpečit přenos dat proti vzniku $t = 3$ nezávislým chybám, při informační rychlosti $R \geq 0{,}5$. Tímto úkolem se zabývá kapitola tři, ve které je detailně popsáno vytvoření generující matice, následné určení kontrolní matice pro dekódování a řešení korekce vzniklých chyb v dekodéru.

Dalším úkolem bylo vypracovat podrobný návrh realizace zde vytvořeného kodeku. Touto problematikou se zaobírá kapitola čtvrtá, ve které je popsán způsob vytvoření kodéru a dekodéru, včetně obvodu realizujícího korekci vzniklých chyb, zastoupeného převodníkem [S]→[E] až po nezbytnou část, kterou je řídící obvod sloužící pro komunikaci zde vytvořeného kodeku s protichybovým systémem, jejímž je tento kodek součástí.

V páté kapitole je realizováno ověření funkčnosti kodeku, kde lze vidět, že navržený kodek splňuje zadáním kladené požadavky.

Poslední kapitola je věnována vytvoření desky plošného spoje, kde lze vidět celé zapojení kodeku a však z důvodu použití freewarového programu pro tvorbu desek plošných obvodů a celkové složitosti obvodu je rozdělena do více desek, které musí být navzájem propojeny. Zde navržené obvody jsou realizovány pro napájecí napětí 5V.

Z návrhu obecného lineárního kódu, je vidět, že náročnost zabezpečení mnohonásobně roste podle počtu chyb, které je kód schopný opravit. Problém u tohoto kódu je pak v obvodu korekce, kdy při návrhu kódu při informační rychlosti $R = 0{,}5$, je počet chybových slov 6, pro variantu opravy jedné chyby, pro variantu opravy dvou chyb, je pak počet chybových slov 78 a při variantě tří chyb je to 1793 chybových slov. Tento jev má potom dopad na velikost dekódovací tabulky potřebnou pro dekódování přijaté zprávy. To vede na procházení více řádků a celkovému zpožďování. U kódování tří chyb je jistou alternativou Golayův kód, který je přímo určen pro zabezpečení zde požadovaného počtu chyb. Výhodou tohoto kódů je nepoužívání dekódovacích tabulek ale výpočet chybového slova za pomoci syndromu a tzv. pomocného slova. Použití Golayova kódu však nebylo předmětem řešení této bakalářské práce.

Výsledky řešení zabezpečení přenosu dat obecnými lineárními blokovými kódy popsané v této práci dokladují splnění zadání bakalářské práci.

LITERATURA

[1] LEE,L.H.CH. Error-Control Block Codes for Communications Engineers. Artech House, Boston,London, ISBN 1-58053-032-X.

[2] HOUGHTON,A. Error Coding for Engineers. Kluwer academic Publishers, Boston, Dordrecht, London. 2001.

[3] ADÁMEK,J. Kódování. Nakladatelství technické literatury, Praha 1989.

[4] NĚMEC, K. *Datová komunikace*. Skriptum. Ústav telekomunikací VUT v Brně, 1998, ISBN 80-7204-088-X.

[5] PACHER J. *Protichybové systémy využívající blokové kódy*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 45 s. Vedoucí bakalářské práce doc. Ing. Karel Němec, CSc.

[6] DUŠEK J. Návrh a implementace opravných kódů pro systém Orpheus. Místo: České vysoké učení technické v Praze. Fakulta elektrotechnická. Studijní program: Informatika a výpočetní technika, 2007. Vedoucí práce byl Ing. Martin Daněk, Ph.D.

[7] SHANNON, C.E. *A Mathematical Theory of Communication* .Bell Syst. tech. J., Vol. 27, No. 3, July 1984 , pp. 379-423, and Vol. 27, No. 4, October 1948, pp. 623-656.

[8] ZEMAN, V. – ŠILHAVÝ, P. – HABR, M. *Datová komunikace počítačová cvičení*. Skriptum. Ústav telekomunikací VUT v Brně

[9] MC9S12XF512 datasheet [online]. Dostupný z WWW: < http://www.freescale.com/ files/microcontrollers/doc/ref_manual/MC9S12XF512V1RM.pdf?fpsp=1 >

[10] FCF4020B datasheet [online]. Dostupný z WWW: <http://www.bucek.name/pdf/4020.pdf>

[11] HCF4081B datasheet [online]. Dostupný z WWW: < http://www.bucek.name/pdf/4081.pdf >

[12] HCF4069UB datasheet [online]. Dostupný z WWW: < http://www.bucek.name/pdf/4069.pdf >

[13] HCF4094B datasheet [online]. Dostupný z WWW: < http://www.bucek.name/pdf/4094.pdf >

[14] HCF4021B datasheet [online]. Dostupný z WWW: < http://www.bucek.name/pdf/4021.pdf >

[15] HCF4070B datasheet [online]. Dostupný z WWW: < http://www.bucek.name/pdf/4070.pdf >

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

d	Hammingova vzdálenost
$d_{\min}$	minimální Hammingova vzdálenost
$P(x)$	polynom nezabezpečené zprávy
$F(x)$	polynom zabezpečené zprávy
$J(x)$	polynom přenesení zprávy
$E(x)$	polynom chybového slova
$S(x)$	polynom syndromu
$R(x)$	polynom opraveného slova
n	délka zabezpečeného bloku
k	délka nezabezpečeného bloku
r	délka zabezpečovacího bloku
$\mathbf{G}$	generující matice
$\mathbf{H}$	kontrolní matice
$\mathbf{B}$	zabezpečovací podmatice
$\mathbf{E}$	jednotková podmatice
w	Hammingova váha