# Digitální modulace a demodulace s podporou hradlovými poli FPGA

Digital modulation and demodulation with support of gate arrays FPGA

Bc. Jiří Světlík

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

# ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Jiří SVĚTLÍK** |
| Osobní číslo: | **A08822** |
| Studijní program: | **N 3902 Inženýrská informatika** |
| Studijní obor: | **Počítačové a komunikační systémy** |
| Téma práce: | **Digitální modulace a demodulace s podporou hradlovými poli FPGA** |

Zásady pro vypracování:

1. **Zpracování rešerše o problematice digitálních modulací PCM včetně zrychlení pomocí adaptivních metod a robustního kódování. (převod lineárního na robustní kódování).**
2. **Rozbor vlastností kódových zabezpečení cyklických kódů pro opravu plánovaného počtu chyb. (kodér systematického cyklického kódu (12,8)-kódu).**
3. **Uspořádání modulátoru a demodulátoru pro digitální modulaci s tvrdým rozhodováním.**
4. **Meggitův dekodér systematického cyklického kódu. ((12,8)-kódu).**
5. **Převod robustního na lineární kódování PCM signálu s korekcí chyby převodu.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:


1. **HRDINA, Zdeněk, VEJRAŽKA, František. Digitální radiová komunikace. 1994. vyd. Praha : Vydavatelství ČVUT, 1994. 257 s. ISBN 80-01-01059-7.**
2. **PROKEŠ, Aleš. Přenos dat. Přenos dat. 2000, 1 2000, č. 1, s. 1-45.**
3. **PRCHAL, J., ŠIMÁK, B.: Digitální zpracování signálů v telekomunikacích, ČVUT Praha, Fakulta elektrotechnická, Praha 2001.**
4. **SEDLÁČEK, M.: Zpracování signálů v měřící technice, ČVUT Praha, Fakulta elektrotechnická, Praha 1993.**
5. **VÝMOLA, Lukáš. Vyhodnocení vlastností A/D převodníků pro digitální zpracování akustických signálů. [s.l.], 2005. 62 s. Vedoucí diplomové práce Vlček Karel, prof. Ing. CSc. .**
6. **Vlček, K.: Komprese a kódová zabezpecení v multimediálních komunikacích, Praha, BEN technická literatura, 2004, ISBN 80-86056-68-6 .**
7. **Sweeney, P.: Error Control Coding: From Theory to Practice, John Wiley & Sons, 2002, ISBN 0-470-84356-X .**
8. **Meggit, J. E.: Error-Correcting Codes for Correcting Bursts of Errors. IBM J. Res. Develop. 4, (1960)**


Vedoucí diplomové práce: **prof. Ing. Karel Vlček, CSc.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **7. června 2010**

Ve Zlíně dne 19. února 2010

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
*děkan*

prof. Ing. Karel Vlček, CSc.
*ředitel ústavu*

## ABSTRAKT

Práce se zabývá digitální modulací a demodulací s podporou hradlovými poli FPGA. V začátku práce je rozebrána digitální modulace PCM včetně zrychlení pomocí adaptivních metod a robustního kódování. Dále je v práci uveden rozbor vlastností kódových zabezpečení cyklických kódů pro opravování plánovaného počtu chyb a rozbor systematického cyklického kódu (12,8). Dále je zde popisováno zapojení kodéru a dekodéru pro systematický cyklický kód (12,8). Následně je tento kodér a dekodér realizovaný v prostředí Quartus II od firmy Altera. V závěru práce je uvedeno ověření funkčnosti kodéru a dekodéru pro kód (12,8).



Klíčová slova: Pulzní kódová modulace, Meggitův dekodér, systematický cyklický kód, digitální modulace, digitální demodulace, FPGA, LSFR.



## ABSTRACT

This work deals with digital modulation and demodulation with the support of gate arrays FPGAs. In the beginning of the work is analyzed PCM digital modulation, including acceleration techniques using adaptive and robust coding. Next part is about characteristics of code security analysis of cyclic codes for renovation of the planned number of mistakes and analysis of the systematic cyclic code (12.8). Next part work is about integration the encoder and decoder for a systematic cyclic code (12.8). After described integration is encoder and decoder implemented in development environment of Quartus II from Altera Company. In conclusion it is shown testing the functionality of the encoder and decoder for the code (12.8).



Keywords: Pulse code modulation, Meggit encoder, systematic cyclic code, digital modulation, digital demodulation, FPGA, LSFR.

Rád bych poděkoval vedoucímu své diplomové práce prof. Ing. Karlu Vlčkovi, CSc. za jeho připomínky, doporučení a rady ohledně vypracování této diplomové práce a také za zapůjčení odborné literatury související s prací.

Dále bych rád poděkoval panu Ing. et Ing. Eriku Královi za cenné rady ohledně programovatelných hradlových polí FPGA i s ukázkami použití a za zapůjčení odborné literatury k těmto polím.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

# OBSAH

# ÚVOD

Jelikož používání kódů je v současné době již nutností v celé řadě různých oborů, výjimkou nejsou ani systematické cyklické kódy, které se v současné době hojně používají v komunikaci pro přenos a zabezpečení video a telefonních signálů. Zpracování tohoto tématu bylo ovlivněno převážně knihou od profesora Vlčka s názvem Komprese a kódová zabezpečení v multimediálních komunikacích, která se zabývá rozborem jednotlivých kódů.

V práci jsou použity operace s kódovými slovy, generujícími polynomy a zbytky po dělení těchto polynomů. Tyto matematické operace jsou nejprve řešeny na úrovni operací s mnohočleny a jejich kořeny a poté simulovány na modelech využívajících zákaznická hradlová pole FPGA.

Návrhy a simulace obvodových řešení jsou realizována v prostředí Quartus II od firmy Altera, která tato vývojové prostředí poskytuje ve verzi Web Edition zcela zdarma. Při volbě vývojového prostředí jsem uvažoval i o největší firmě poskytující zákaznická hradlová pole FPGA, a to o firmě Xillinx, ale jejich vývojové prostředí již v bezplatné verzi WebPack má do značné míry omezené důležité vizuální a vývojové prostředky, proto jsem se rozhodnul využít prostředí od firmy Altera.

Tato práce poskytuje základní informace o kódování systematických cyklických BCH kódů a jejich obvodové a simulační řešení prostřednictvím zákaznických hradlových polí FPGA. Dále je v práci pojednáno o dekódování systematických cyklických kódů s využitím Meggitova dekodéru. V závěru práce je uveden model zapojení kodéru a dekodéru realizovaný ve vývojovém prostředí Quartus II. Zapojení funkčních modelů je ověřeno na několika informačních slovech, které byly předem vypočítány (v prostředí Mathematica) a poté ověřeny průchodem přes tyto modely kodéru a dekodéru.

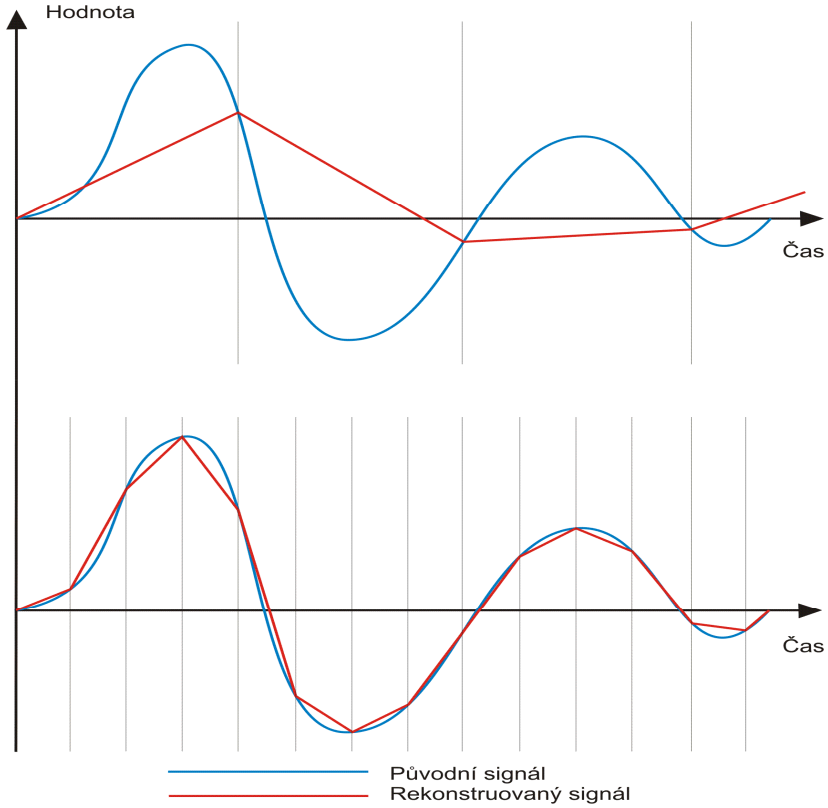
# I. TEORETICKÁ ČÁST

# 1 DIGITÁLNÍ MODULACE PCM

Modulace PCM neboli Pulse-Code Modulation se používá především pro převod zvukového analogového signálu na signál digitální. Podstatou PCM jsou tři základní operace (kvantování, vzorkování a kódování). Principielně funguje PCM tak, že A/D převodníkem pravidelně odečítají hodnoty vzorku a ty jsou ukládány v binárních hodnotách. Jednotlivým hodnotám vzorků jsou přiřazovány určité tzv. kvantizační hladiny a těmto kvantizačním hladinám nebo také úrovním odpovídá určitá kódová kombinace nul a jedniček.

## 1.1 Vzorkování

Vzorkování signálu probíhá jako první část při PCM modulaci. Nejprve se musí na signálu provést vzorkování, to znamená diskretizovat zpracovávaný signál v časové oblasti. Vzorkování se provádí tak, že se průběh signálu rozdělí do jednotlivých úseků a v těchto úsecích se odeberou hodnoty. Tímto procesem ztratíme mnoho detailů o průběhu analogového signálu, ovšem na druhou stranu můžeme signál lépe zpracovat a uchovat v digitální technice. Množství vzorků v závislosti na průběhu signálu nám určuje Shannonův teorém někdy také nazývaný jako Shannonův-Nyquistův-Kotělnikův teorém. Pokud bychom vzorkovali méně, než podle tohoto teorému, ztráceli bychom důležité detaily signálu a tento signál by se znehodnocoval. V následujícím obrázku jsou uvedeny dva případy vzorkování. V prvním případě je vidět malý počet vzorků, u kterého dochází ke zkreslování signálu. V druhém případě je počet vzorků větší, což vede k věrnější rekonstrukci signálu. Aby nebyl počet vzorků odhadován, vypočítává se pomocí již dříve zmíněného Shannonova-Nyquistova-Kotělníkova teorému.




Obr. č. 1 – Špatné a dobré vzorkování

### 1.1.1 Shannonův-Nyquistův-Kotělníkův teorém

Shanonnův-Nyquistův-Kotělníkův teorém říká, že „Přesná rekonstrukce spojitého, frekvenčně omezeného signálu z jeho vzorků je možná tehdy, pokud byl vzorkován frekvencí alespoň dvakrát vyšší, než je maximální frekvence rekonstruovaného signálu.“ [1] V praxi se volí vzorkovací frekvence dvaapůlkrát vyšší, než je maximální frekvence rekonstruovaného signálu, aby byla k dispozici menší rezerva. V telekomunikacích je to například 8 kHz, neboť přenášený signál ve standardním telefonním pásmu je od 0,3 – 3,4 kHz. Pokud by se při vzorkování nedodržela dvakrát větší vzorkovací frekvence, mohlo by dojít k aliasingu tedy k výraznému zkreslení signálu. Následující vzoreček vyjadřuje Shannonův teorém, kde $f_v$ je vzorkovací frekvence a $f_{max}$ je největší harmonická přenášená frekvence.

$$f_v \geq 2f_{max}$$

## 1.2 Kvantování

Jestliže je vzorkování diskretizace signálu v čase, potom kvantování je diskretizace hodnoty signálu v jeho amplitudě. Tudíž se může říci, že kvantovaný signál na výstupu obsahuje jen určitý počet hodnot. Signál lze upravit na potřebné množství úrovní několika způsoby: oříznutím signálu zespod, oříznutí signálu shora a zaokrouhlováním signálu.

Signál vycházející z kvantizátoru lze rozdělit na vstupní signál do kvantizátoru a na chyby vytvořené kvantizátorem. Tyto chyby jsou nazývány jako kvantizační šum a jsou tím menší, čím je více kvantizačních hladin. Obvykle se pro kvantování používá 8 nebo 16 bitů (256 respektive 65536 možných hodnot). Kvantováním signálu vždy ztrácíme informace. Mimo kvantizační hladiny se uvádí i pojem rozhodovací hladiny. Těmito rozhodovacími hladinami jsou myšleny úrovně, na kterých dochází k rozhodování, pro kterou kvantovací hladinu bude vstupní signál zaokrouhlen.

### 1.2.1 Lineární kvantování

Kvantování lze rozdělit podle způsobu kvantizace. Při lineárním kvantování je volen rozestup kvantizačních hladin rovnoměrně. To znamená, že jednotlivé kvantizační hladiny jsou od sebe vzdáleny konstantně o určitou délku. Tento způsob kvantizace signálu je nenáročný na realizaci a využívá se ho tam, kde je signál rovnoměrně rozdělen.

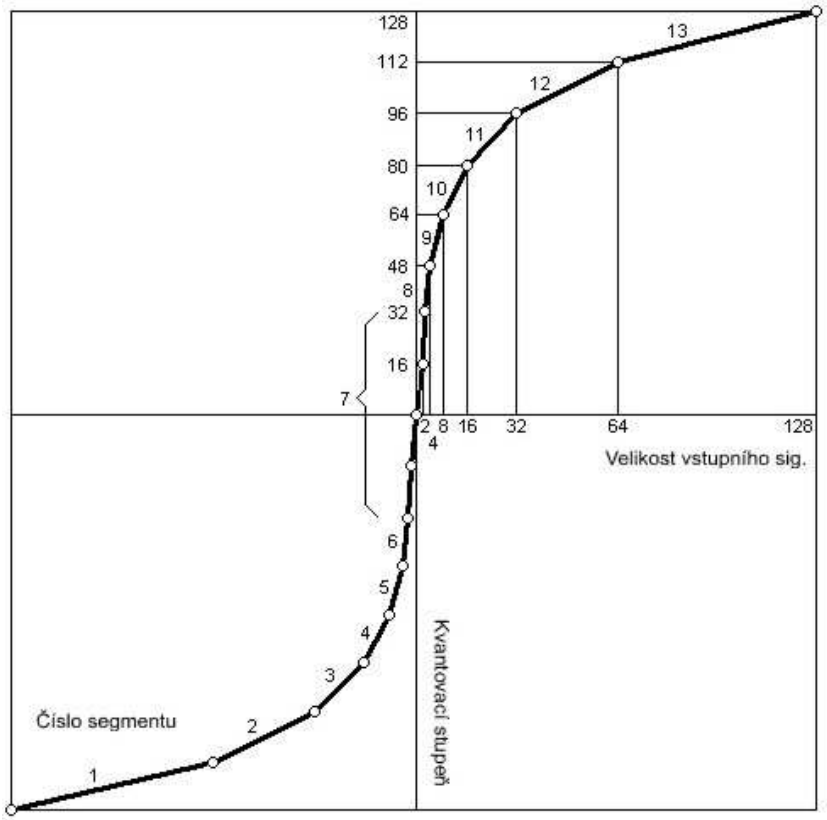
### 1.2.2 Nelineární - Robustní kvantování

Naproti tomu u robustního kvantování je způsob kvanitzace zcela odlišný. Při tomto způsobu kvantizace je volen rozestup kvantizačních hladin v závislosti na pozdějším detailním popsání určitých částí signálu při jeho rekonstrukci. Není rozdělen rovnoměrně a v určitých částech signálu může být počet kvantizačních hladin hustší, nebo řidší. Tento způsob kvantizace signálů se využívá tam, kde je například u rychlých změn signálu potřeba hustší pokrytí více body. Nevýhoda tohoto způsobu kvantování je v náročnější realizaci, než je realizace lineárního způsobu kvantování. Využívá se především v telefonii, protože se umí vypořádat se širokým rozsahem dynamiky vstupních signálů.

## 1.3 Kódování

Kódování signálu se používá pro přenos signálu přenosovým prostředím. Z pohledu počtů úrovní signálu lze kódování rozdělit na: Unipolární signály, Polární signály, Bipolární signály a vícestavové signály. Nejčastěji používaná kódování jsou: NRZ, RZ, NRZI, PSK, DPSK, PCM.

Pro přenos řečového signálu se v telekomunikační technice používá 13-bitový vzorek s opakovacím kmitočtem 8kHz. Tento vzorek je však příliš dlouhý pro přenos sériovým vedením, a proto je zkracován na osmibitový vzorek. Podle komise CCITT (Comité Consultatif International Téléphonique et Télégraphique) bylo pro Evropu doporučeno aby A/D převodník byl 13-bitový, a aby po něm následovala komprese vzorků podle takzvané A-křivky pro Evropu a µ-křivky pro USA a Japonsko. Tato křivka je znázorněna na následujícím obrázku.




Obr. č. 2 – A-křivka pro převedení komprese v Evropě

# 2 ZRYCHLENÍ MODULACE POMOCÍ ADAPTIVNÍCH METOD

Zrychlením modulace pomocí adaptivních metod je myšleno zrychlení přenosu kódovaného signálu prostředím. To se děje zmenšením počtu přenášených bitů přes přenosové prostředí. K tomuto účelu se používají především dvě metody, a to: Diferenciální PCM a Adaptivní diferenciální PCM.

## 2.1 Diferenciální pulzní kódová modulace

Systémy, které používají Diferenciální PCM, kódují rozdíly mezi okamžitou hodnotou vzorku signálu v daném vzorkovacím okamžiku a hodnotou predikovanou z předchozích vzorků. Tím dochází k redukci počtu bitů, zrychlení přenosové rychlosti a maximalizaci množství přenášených dat.

## 2.2 Adaptivní diferenciální pulzní kódová modulace

Adaptivní diferenciální PCM vychází z DPCM. Tato modulace je vylepšena o adaptivní změny v počtu kvantizačních hladin. To znamená, že se adaptivně přizpůsobuje statistickým parametrům řeči. Tím je způsobena ještě větší redukce počtu bitů.

# 3 PŘEVOD LIENÁRNÍHO NA ROBUSTNÍ KÓDOVÁNÍ

Převodem lineárního na robustní kódování je myšleno zkrácení přenášeného slova. Všeobecně platí, že čím je menší počet přenášených znaků, tím je menší pravděpodobnost výskytu chyby ve zprávě. Z A-křivky (obr. č. 1) vidíme, že třináctibitové slovo je zkráceno na 8 bitů.

## 3.1 Princip převodu 13-ti bitového slova na 8-mi bitové slovo

Vektor $x = \{P\ X_{11}\ X_{10}\ X_9\ X_8\ X_7\ X_6\ X_5\ X_4\ X_3\ X_2\ X_1\ X_0\ \}$ představuje třináctibitový vzorek získaný lineárním A/D převodníkem. Bit P je znaménkový bit (většinou je používán ve smyslu: je-li P=1, je polarita kladná, je-li P=0, je polarita záporná). Při kompresi podle A-křivky je tento třináctibitový vektor zkrácen na osmibitový vektor ve tvaru: $x = \{P\ s_2\ s_1\ s_0\ q_3\ q_2\ q_1\ q_0\}$. Bit P je opět vyjádřením polarity. Je tedy při kompresi beze změny přenesen do vektoru y. Bity $s_0$, $s_1$ a $s_2$ jsou vytvářeny jako binární vyjádření čísla 7 mínus počet nulových bitů obsažených ve vektoru x za znaménkovým bitem P.[2] Převod při kompresi ukazuje tabulka číslo 1.

| | **třináctibitové slovo** | | | | | | | | | | | | | **osmibitové slovo** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| | | **$X_{11}$** | **$X_{10}$** | **$X_9$** | **$X_8$** | **$X_7$** | **$X_6$** | **$X_5$** | **$X_4$** | **$X_3$** | **$X_2$** | **$X_1$** | **$X_0$** | | **$S_2$** | **$S_1$** | **$S_0$** | **$q_3$** | **$q_2$** | **$q_1$** | **$q_0$** |
| | P | 0 | 0 | 0 | 0 | 0 | 0 | 0 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | P | 0 | 0 | 0 | $q_3$ | $q_2$ | $q_1$ | $q_0$ |
| | P | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | P | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ |
| | P | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | - | P | 0 | 1 | 0 | q3 | q2 | q1 | $q_0$ |
| | P | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | - | - | P | 0 | 1 | 1 | q3 | q2 | q1 | $q_0$ |
| | P | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | - | - | - | P | 1 | 0 | 0 | q3 | q2 | q1 | $q_0$ |
| | P | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | - | - | - | - | P | 1 | 0 | 1 | q3 | q2 | q1 | $q_0$ |
| | P | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | - | - | - | - | - | P | 1 | 1 | 0 | q3 | q2 | q1 | $q_0$ |
| | P | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | - | - | - | - | - | - | - | P | 1 | 1 | 1 | q3 | q2 | q1 | $q_0$ |

Tab. č. 1. – Tabulka převodu třináctibitového slova na osmibitové

## 3.2 Princip převodu 8-mi bitového slova na 13-ti bitové

Při expanzi je ve třináctibitovém slově doplněn nejvýznamnější ztracený bit jedničkou a zbývající ztracené bity nulami. Tímto způsobem je minimalizována chyba, která ztrátou vzniká, protože hodnota, doplněná ve ztracených bitech, je polovinou maximální možné ztracené hodnoty. Komprese zprávy prováděná tímto způsobem je založena na snížení informačního obsahu vzorků pro největší amplitudy (převod je vlastně pouze šestibitový) a zachování plného informačního obsahu nejmenších amplitud signálu (převod je třináctibitový s chybou o hodnotě poloviny nejméně významného řádu ve slově). Výsledkem je zkrácení zprávy na 8 bitů z původních 13bitů délky při zachování přesnosti nejmenších amplitud. [3] Převod osmibitového slova y na třináctibitové slovo x je popsán v tabulce číslo 2.

| | **osmibitové slovo** | | | | | | | | **třináctibitové slovo** | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| | | **$S_2$** | **$S_1$** | **$S_0$** | **$q_3$** | **$q_2$** | **$q_1$** | **$q_0$** | | **$X_{11}$** | **$X_{10}$** | **$X_9$** | **$X_8$** | **$X_7$** | **$X_6$** | **$X_5$** | **$X_4$** | **$X_3$** | **$X_2$** | **$X_1$** | **$X_0$** |
| | P | 0 | 0 | 0 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | P | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 |
| | P | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | P | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 |
| | P | 0 | 1 | 0 | q3 | q2 | q1 | $q_0$ | P | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | 0 |
| | P | 0 | 1 | 1 | q3 | q2 | q1 | $q_0$ | P | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | 0 | 0 |
| | P | 1 | 0 | 0 | q3 | q2 | q1 | $q_0$ | P | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | 0 | 0 | 0 |
| | P | 1 | 0 | 1 | q3 | q2 | q1 | $q_0$ | P | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | 0 | 0 | 0 | 0 |
| | P | 1 | 1 | 0 | q3 | q2 | q1 | $q_0$ | P | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | 0 | 0 | 0 | 0 | 0 |
| | P | 1 | 1 | 1 | q3 | q2 | q1 | $q_0$ | P | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

Tab. č. 2 – Tabulka převodu osmibitového slova na třináctibitové

# 4 ROZBOR VLASTNOSTÍ KÓDOVÝCH ZABEZPEČENÍ CYKLICKÝCH KÓDŮ PRO OPRAVU PLÁNOVANÉHO POČTU CHYB

## 4.1 Cyklické kódy

Cyklický kód je lineární kód, který se užívá k takzvanému cyklickému posunu. Lineární kód se může nazývat cyklickým kódem, jestliže pro každé kódové slovo $(a_0, a_1, a_2, \ldots\ldots a_{n-1})$ je také slovo $(a_{n-1}, a_0, a_1, a_2, \ldots\ldots a_{n-2})$ kódovým slovem. To znamená, že cyklickým posuvem kódového slova vzniká zase kódové slovo. Tato slova cyklického kódu délky n se zapisují ve tvaru formálních polynomů stupně menšího než „n“ s využitím izomorfismu

$$(a_0, a_1, a_2, \ldots\ldots a_{n-1}) \rightarrow\ a_0\ + a_1 x + \cdots + a_{n-1} x^{n-1}\ .$$

Každý cyklický kód je jednoznačně určen svým generujícím polynomem g(x). Výsledný cyklický kód se skládá právě z násobků generujícího polynomu g(x). Tento generující polynom g(x) dělí polynom $x^n - 1$ bezezbytku.

Podíl $h(x) = \frac{x^n - 1}{g(x)}$ představuje kontrolní mnohočlen, nebo také kontrolní polynom cyklického kódu. Má-li cyklický kód opravovat jednonásobné chyby, musí být g(x) alespoň trojčlenem a z množiny násobků g(x) je nutno vyloučit všechny dvojčleny, aby minimální kódová vzdálenost byla $d_{min} > 2$. [4]

## 4.2 Kód (12,8)

Je nutné zdůraznit, že kód (12,8) je redundantní cyklický kód, který vychází z cyklického kódu (15,11), kde jsou zanedbané (vynechané) ty nejvyšší informační bity. Ovšem mohou být zanedbané (vynechané) jen v případě, že jsou nulové. Zkrácením informačních bitů se ovšem nezkracují kontrolní bity. Tyto zůstávají stejné, a tedy 4 bitová redundantní část se nemění. Pokud je posledním patnáctým bitem znaménko, musí se zachovat a nesmí se vynechat spolu s nejvyššími bity. Zpracovávání takto dlouhých slov se znaménkem je realizováno tak, že znaménko se zpracovává zvlášť. To znamená, že zpracování kódu (15,8) se dá rozdělit na tři části: zpracování znaménkového bitu, zpracování informačních bitů a zpracování kontrolních (redundantních) bitů.

## 4.3 Realizace kódu (12,8) vynecháním některých kódových slov

Jeden ze způsobu realizace kódu (12,8) je takový, že ze všech kódových slov vybereme jen ta, která mají délku kódového slova nejvýše 12 bitů. Použitý kód je systematický s oddělenou redundantní částí, a proto lze tato slova snadno vyčlenit. V kontrolní (redundantní) části nemusí být všechny čtyři kontrolní bity vždy využity, ale jejich počet nesmí být změněn. V některých případech se může stát, že i nejvyšší kontrolní bit je nenulový, a proto jej nesmíme vynechat. V následující tabulce jsou červeně vyznačena nevybraná (vynechaná) slova.

| | patnáctibitové slovo | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | $X_{14}$ | $X_{13}$ | $X_{12}$ | $X_{11}$ | $X_{10}$ | $X_9$ | $X_8$ | $X_7$ | $X_6$ | $X_5$ | $X_4$ | $X_3$ | $X_2$ | $X_1$ | $X_0$ |
| | P | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 |
| | P | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - |
| | P | 0 | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - |
| | P | 0 | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - | - |
| | P | 0 | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - | - | - |
| | P | 0 | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - | - | - | - |
| | P | 0 | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - | - | - | - | - |
| | P | 0 | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - | - | - | - | - | - |
| | P | 1 | $q_3$ | $q_2$ | $q_1$ | $q_0$ | 1 | - | - | - | - | - | - | - | - |

Tab. č. 3. – Tabulka použitých kódových slov

# 5 USPOŘÁDÁNÍ MODULÁTORU A DEMODULÁTORU PRO DIG. MODULACI S TVRDÝM ROZHODOVÁNÍM

## 5.1 Modulace a demodulace

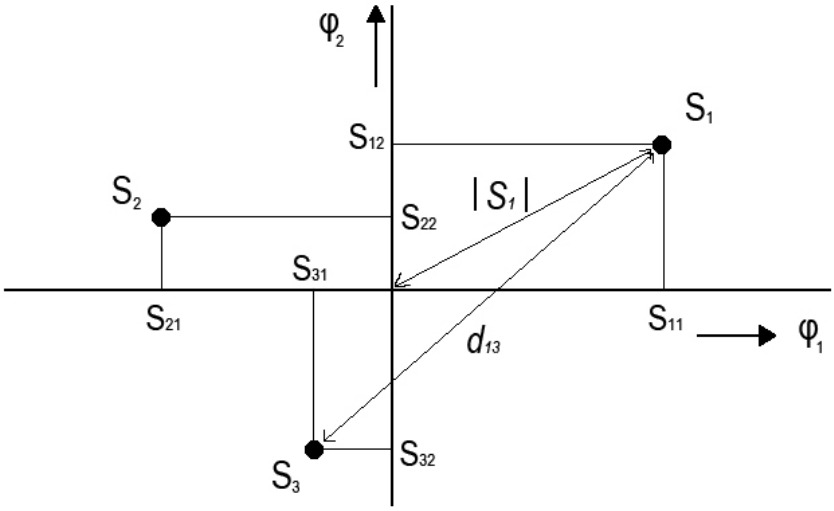
Informaci je v digitální technice možné přenášet od modulátoru k demodulátoru mnoha způsoby a tím i využívat různých modulací. Zvolená modulace PCM se využívá pro kabelové vedení z důvodu nízkého rušení. Pokud bychom chtěli přenášet informaci prostřednictvím vzduchu, bylo by vhodnější použít modulaci PSK (Phase shift keying). Přenášení signálu ve formě spojité veličiny vyžaduje na vysílací straně převést diskrétní znaky symbolů na spojitý signál (tomuto převodu říkáme modulace) a po přenosu signálu spojitým sdělovacím kanálem převést spojitý přijatý signál zpět na znaky (tomuto převodu říkáme demodulace).

Pokud budeme předpokládat, že binární symboly vstupující do modulátoru nabývají obou možných znaků stejně pravděpodobně a nezávisle. Tomuto říkáme, že kódováním se zdroj zpráv stal optimální. V modulátoru můžeme každému znaku přiřadit jeden ze dvou signálů a poté mluvíme o binární (dvoustavové) modulaci. Obecně však můžeme každé k-tici binárních znaků přiřadit jeden z $M = 2^k$ signálů a poté mluvíme o M-ární (M-stavové) modulaci. [5]

## 5.2 Signálový prostor

Popis způsobů modulace a jejich vlastností výrazně ulehčuje zobrazení modulovaného signálu jako bod v signálovém prostoru. Modulovaný signál je plně charakterizován koeficienty rozkladu do báze a lze si jej představit jako bod v prostoru se souřadnicemi $S_{ik}$ na souřadných osách $\varphi_k, k = 1, \dots, n$.

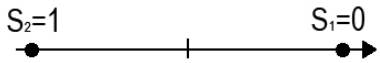
Čím více jsou od sebe signálové body vzdálenější, tím více je větší energie rozdílu jejich signálů. Tato energie je měřítkem odlišnosti signálů a proto platí, že čím více jsou vzdálenější body dvou signálů, tím více budou tyto signály vzájemně odlišitelné na přijímací straně. [6] Signálový prostor se třemi body signálů je znázorněn v následujícím obrázku.




Obr. č. 3 – Zobrazení modulovaných signálů v signálovém prostoru

## 5.3 Rozložení signálových bodů v signálovém prostoru pro dvoustavovou PCM

PCM je modulace používající binárních symbolů. Tedy nul a jedniček. Rozložení těchto symbolů bude souměrné kolem osy Y a každý z nich bude ležet na jedné straně. Na následujícím obrázku je znázorněno rozložení těchto bodů spolu s přiřazením k přenášeným binárním znakům.




Obr. č. 4. – Rozložení signálových bodů pro PCM

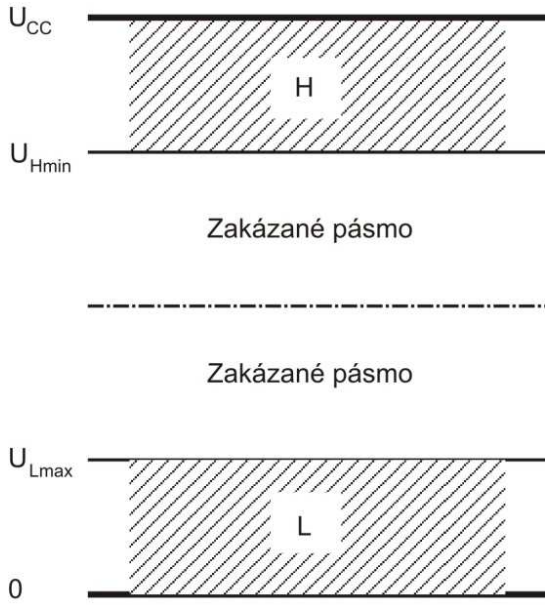
Tyto dvoustavové signály snižují nároky na výrobní náklady, proto je možné při výrobě součástek pro dvoustavové signály zavést hromadnou a levnou výrobu. Když uvažujeme, že číslicové součástky jsou napájeny kladným napětím $+U_{CC}$ jedna hodnota bude vyjádřena nižším napětím a druhá vyšším. Tyto hodnoty mohou být označeny jako ′0′

a ′1′ v souladu se značením v Booleově algebře. Jejich přiřazení k vyššímu či nižšímu napětí se může realizovat dvojím způsobem.

1. Pozitivní logikou – v tomto případě odpovídá nižší napětí hodnotě „0“ a vyšší napětí hodnotě „1“.

2. Negativní logikou – v tomto případě odpovídá vyšší napětí hodnotě „0“ a nižší napětí hodnotě „1“.

Zmírnění nároků na tolerance napětí je dosaženo tak, že logickým hodnotám jsou přiřazena celá pásma napětí, jak je vidět v následujícím obrázku (obr. č. 5).



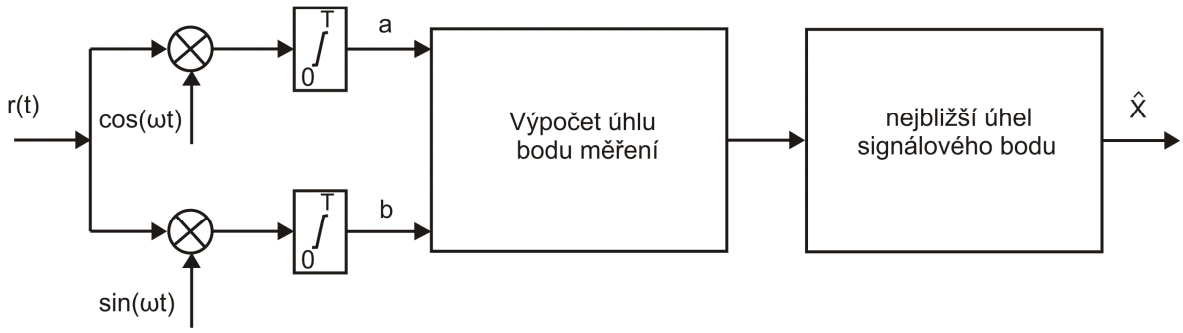

Obr. č. 5 – Pásma napětí u číslicových signálů

Z obrázku je patrné, že existuje pásmo nižších napětí L ( Low) a pásmo vyšších napětí H (High). Mezní hodnoty jsou tady zastoupeny jako $U_{Lmax}$ a $U_{Hmin}$. Pokud bychom přemýšleli v pozitivní logice, tak by veškeré hodnoty napětí menší než $U_{Lmax}$odpovídaly hodnotě nula a všechny hodnoty napětí větší než $U_{Hmin}$ by odpovídaly hodnotě jedna. Mezi těmito mezemi je zakázané pásmo, přes které signál přechází jen při změně stavu obvodu. [7]

Za ideálních podmínek, tj. hodnoty uvnitř přípustných pásem a při nekonečně rychlých přechodech mezi nimi, lze místo s hodnotami napětí pracovat s logickými hodnotami „0“ a „1“. Kdybychom tedy uvažovali, že máme ideální podmínky, můžeme používat Booleovu algebru při návrhu systému.

## 5.4 Uspořádání demodulátoru pro PCM

Uspořádání demodulátoru pro modulaci PCM vychází z obecného zapojení demodulátoru pro demodulaci signálových bodů v signálovém prostoru. Vzhledem ke stejné energii modulovaných signálů je demodulátor založen na hledání signálových bodů s nejmenší úhlovou vzdáleností od bodu měření. Schéma demodulátoru je znázorněno na následujícím obrázku. [7]




Obr. č. 6. – Demodulátor pro PCM

Výsledek po demodulaci nám říká, jestli signálový bod je demodulovaný správně, či nikoliv. Zapojení demodulátoru je stejné jako pro demodulování signálových bodů PSK s tím rozdílem, že pro binární zpracování PCM se jen zvětší úhel v signálovém prostoru.

## 5.5 Rozdíl mezi měkkým a tvrdým rozhodováním

Při modulaci se sinusovka modulovaného signálu přepóluje na zápornou a tím dojde k modulování nosného signálu. Metodiku demodulování lze obecně rozdělit na vyhodnocování s tvrdým rozhodováním a na vyhodnocování s měkkým rozhodováním. U tvrdého rozhodování o výstupním symbolu rozhoduje četnost výskytu jednotlivých symbolů na více dekodérech. Zatímco u měkkého rozhodování probíhá výpočet, který vrací rekurentní vztah pro další zpracování. Při měkkém rozhodování je brána v úvahu také chyba příslušného výstupního vzorku, jemuž se přiřazuje nejbližší symbolový prvek. Proto měkké rozhodování je přesnější, trvá déle a u více modulačních funkcí přináší přesnější výsledky. Metodika měkkého rozhodování je poměrně rozsáhlá, je závislá na mnoha faktorech (modulace, filtrace apod.) a je spíše námětem pro samostatnou práci.

# 6 MEGGITŮV DEKODÉR SYSTEMATICKÉHO CYKLICKÉHO KÓDU (12,8)-KÓDU

## 6.1 Popis kódů pomocí mnohočlenů

U zabezpečovacích kódů je snaha dosáhnout toho, aby poměr počtu kontrolních symbolů k celkovému počtu symbolů ve slově byl co nejmenší. Tato podmínka je snadněji dosažitelná při co největším počtu symbolů ve slově. Proto jsou aplikovány takzvané (n,k)-kódy s co největším n. Při implementaci kódových algoritmů popsaných maticemi to vede ke složité obvodové realizaci.

Úsilí o nalezení kódových systémů pro kódy s dlouhými kódovými slovy, které by byly realizovány s co nejmenším počtem obvodových prvků, vedlo ke studiu takzvaných Cyklických kódů. V komunikační technice je zpracování zpráv zpravidla sériové (zpráva je zpracovávána jako posloupnost symbolů). A aplikace Cyklických kódů u tohoto zpracování zpráv je výhodnější.

Cyklické kódy umožňují nejúspornější řešení implementační řešení kódového systému. Pro všechny cyklické, blokové kódy je potřeba teorie cyklických kódů, která je založena na výpočetních operacích s mnohočleny a jejich kořeny. [8]

## 6.2 Vytváření systematických cyklických kódů

Systematické kódování je prováděno operací dělení mnohočlenů. Mnohočlen čteme od nejvyššího stupně k nejnižšímu (u nesystematického kódování je to naopak). Následuje použití informačních bitů jako koeficientů nejvyššího stupně:

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

Kódování se dosáhne dělením informačního mnohočlenu u(x) generujícím mnohočlenem g(x). Při dělení je vypočten mnohočlen q(x), který nebude použitý a zbytek po dělení r(x):

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

Kódové slovo potom představuje mnohočlen u(x) + r(x). Jedná se o kódové slovo, které je násobkem libovolného mnohočlenu s generujícím mnohočlenem g(x). Platí:

$$u(x) + r(x) = q(x).g(x)$$

Při vytváření systematického cyklického kódu s generujícím mnohočlenem $g(x) = x^4 + x^3 + 1$ informační bity 01000100 vyjadřuje mnohočlen $u(x) = x^{10} + x^6$. Dělení prováděné generujícím mnohočlenem $g(x)$ poté vypadá následovně:

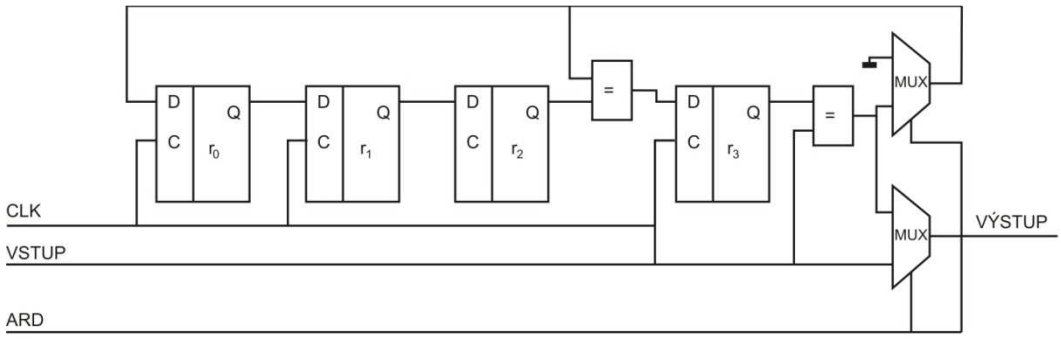
$$(x^{10} + x^6) : (x^4 + x^3 + 1) = x^6 + x^5 + x^4 + x^3 + x^2 + 1$$

$$r(x) = x^2 + 1$$

Zbytek po dělení je tedy $r(x) = x^2 + 1$ a vyslán bude mnohočlen ve tvaru kódového slova $w(x) = u(x) - r(x) = x^{10} + x^6 + x^2 + 1$, což odpovídá slovu 010001000101. To znamená, že mnohočleny u(x) a r(x) se nebudou sčítat a tím je zaručena oddělená informační část od zabezpečovací části. [9]

### 6.2.1 Funkce kodéru cyklického systematického kódu

Kodér pracuje v několika krocích, a to tak, že při prvních jedenácti posuvech jsou vyslány informační bity $u_{14}, u_{13}, u_{12} \ldots\ldots\ldots u_5, u_4$ a obvod pro dělení mnohočlenu vytváří zbytek po dělení a to bity $r_3, r_2, r_1, r_0$. Při těchto prvních jedenácti krocích jsou multiplexory adresovány tak, že na jejich výstupech se objeví hodnota logických signálů přivedených na vstup (původní informační slovo u(x)). Poté je ve čtyřech krocích na výstup kodéru odeslán zbytek po dělení, který se vytvořil na jednotlivých klopných obvodech. Na vstupu kodéru musím být na začátku nulová hodnota, aby nebyl výsledek ovlivněn předchozí hodnotou. Tento zbytek je připojen k informačním bitům pomocí multiplexorů, které jsou v určitou dobu správně sepnuty. Zapojení kodéru je znázorněno na následujícím obrázku. [10]




Obr. č. 7 – Kodér cyklického systematického kódu

## 6.3 Dekódování systematických cyklických kódů

Při dekódování cyklických kódů se používá výpočet syndromu v podobě mnohočlenu a to tak, že:

Nechť K je cyklický kód o délce „n“ s generujícím mnohočlenem g(x). Mnohočlen s(x) nazýváme syndrom přijatého slova w(x) o délce „n“, jestliže s(x) vznikl jako zbytek po dělení mnohočlenu w(x) mnohočlenem g(x). Za pozornost stojí, že přijaté slovo popsané mnohočlenem w(x) má stejný syndrom jako reprezentant chyby popsaný mnohočlenem e(x). Vysláno bylo totiž kódové slovo, které má tvar q(x)g(x) a přijaté bylo slovo w(x)=q(x)g(x)+e(x). Zbytek po dělení rozdílu mnohočlenů w(x)-e(x) mnohočlenem g(x) musí být tedy nulový, neboť u binárních kódů se jedná o lineární operaci a platí tedy princip superpozice. [10]

Nechť K je Hammingův kód délky 12 s generujícím mnohočlenem $g(x) = 1 + x^3 + x^4$. Syndrom s(x) přijatého slova w=010001010101 vypočítáme jako zbytek po dělení:

$$(x^{10} + x^6 + x^4 + x^2 + 1) : (x^4 + x^3 + 1)$$

Z toho vyplývá, že při nalezení syndromu u Hemmingova kódu, dělením přijatého slova w(x) generujícím polynomem g(x), jsme schopni opravit jednu chybu. Nalezením syndromu v následující tabulce určíme, ve kterém bitu se nalézá chyba. V tomto případě je zbytek po dělení $r(x) = x^3 + 1$.

| Chybný bit | Syndrom | | | |
|---|---|---|---|---|
| 0 | 0 | | | |
| 1 | 1 | | | |
| x | | x | | |
| $x^2$ | | | $x^2$ | |
| $x^3$ | | | | $x^3$ |
| $x^4$ | 1 | | | $+ x^3$ |
| $x^5$ | 1 | $+ x$ | | $+ x^3$ |
| $x^6$ | 1 | $+ x$ | $+ x^2$ | $+ x^3$ |
| $x^7$ | 1 | $+ x$ | $+ x^2$ | |
| $x^8$ | | x | $+ x^2$ | $+ x^3$ |
| $x^9$ | 1 | | $+ x^2$ | |
| $x^{10}$ | | x | | $+ x^3$ |
| $x^{11}$ | 1 | | $+ x^2$ | $+ x^3$ |

Tab. č. 4 – Tabulka pro nalezení syndromu

Nalezený syndromový mnohočlen tedy odpovídá poloze reprezentanta chyby $e(x) = x^4$. Přijaté slovo se tedy opraví na kódové slovo podle vztahu:

$$w(x) = w(x) - e(x) = 010001010101 - 000000010000$$

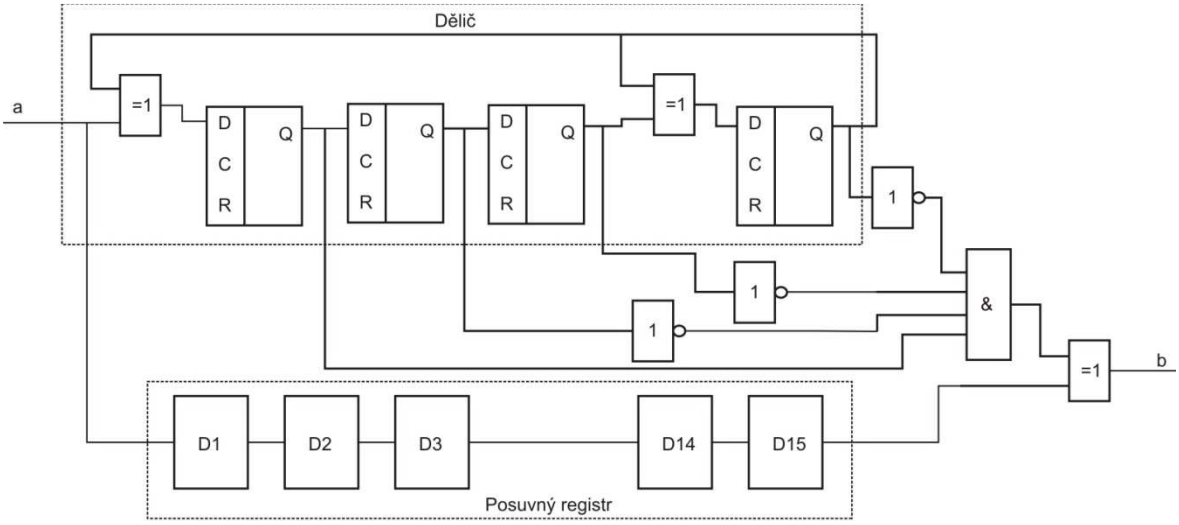
$$w(x) = 010001000101$$

## 6.4 Meggitův dekodér

Nejsložitější částí dekódování pomocí syndromu je vyhledávání a výpis chybových mnohočlenů pro každé přijaté slovo. Postup, který popsal Meggit v roce 1960, nevyžaduje, aby byly známy všechny syndromy až do stupně n-1 hned na začátku. Výpočet reprezentanta chyby (polohy špatného bitu) z mnohočlenu syndromu r(x) bude prováděn v každém kroku vždy pro jedno kódové slovo. Z přijatého slova bude vypočten mnohočlen r(x) pomocí obvodu, který provádí dělení mnohočlenů. Syndromy všech cyklických posuvů jsou generovány tímtéž obvodem.

Princip metody dekódování, objevený v roce 1960 Meggitem, je založen na tom, že syndrom přijatého slova s chybou a syndrom samotného chybového slova mají ekvivalentní zbytek, jsou-li děleny generujícím mnohočlenem. Výpočet syndromu není nutné opakovat pro každý posuv kódového slova: vypočten je syndrom z přijatého slova w(x) a cyklickým posuvem v obvodu pro dělení mnohočlenem g(x) obdržíme syndromy cyklického posuvu ve slově w(x). [11]

Pro opravu příslušeného bitu kódového slova w(x) slouží obvod sestavený ze tří invertorů a jednoho hradla AND se čtyřmi vstupy. Pokud bude spočítán syndrom pro opravu a na výstupu tohoto hradla AND bude "1", tak tato logická hodnota pak provádí opravu příslušného bitu v přijatém slově w(x) pomocí hradla s funkcí EX-OR. Toto zapojení je znázorněno na následujícím blokovém schématu.




Obr. č. 8. – Skupinové schéma dekodéru (15,11)-kódu

Na obrázku číslo 8 je vidět, zapojení obvodu pro dekódování cyklického systematického kódu (15,11) s generujícím polynomem $g(x) = x^4 + x^3 + 1$. V horní části ohraničené a popsané jako dělič je vidět obvod LFSR, který slouží na vyhodnocování výpočtu syndromu dělení mnohočlenem. Ve spodní části obvodu je ohraničen posuvný registr, který je tvořen patnácti klopnými obvody a v prostřední části je umístěno hradlo AND a tři invertory, protože oprava zde nastává jen při syndromu 1000.

Při použití stávajícího schématu pro dekódování (15,11)-kódu je možné tento dekodér použít i pro (12,8)-kód s tím, že zbývající bity budou doplněny nulami. Takto použitý kód bude využívat všechny registry beze změny dekodéru. Ovšem při výpočtu nemusí být použity všechny syndromy. Toto by bylo zbytečné.

# 7 PŘEVOD ROBUSTNÍHO NA LINEÁRNÍ KÓDOVÁNÍ PCM SIGNÁLU S KOREKCÍ CHYBY PŘEVODU.

## 7.1 Převode robustního na lineární kódování PCM signálu

Tímto převodem je myšleno zkrácení patnáctibitové kódu na kód s menším počtem bitů. Tento postup je znázorněn v tabulce číslo jedna uvedené výše. Komprese zprávy prováděná tímto způsobem je založena na snížení informačního obsahu vzorků v největších amplitudách.

## 7.2 Korekce chyby převodu

Korekcí chyby při převodu je myšleno to, že ve kterémkoliv přenášeném bitu může nastat chyba. Úkolem je tuto chybu rozpoznat a opravit. Pro zvolený kód (12,8) bude schopnost rozpoznání dvou chyb a možnost opravy jedné chyby v jednom bitu. Tato chyba se bude rozpoznávat pomocí syndromu, jehož výpočet je popsaný výše. V tabulce číslo pět je znázorněno přijaté slovo bez chyby a přijaté slovo s chybou. Pokud je slovo přijato správně, je v následujících opravných 15 krocích nezměněno (Meggitův dekodér může zachovat všech 15 kroků s tím vědomím, že při použití kódu (12,8) budou tři bity nulové). Pokud je ovšem přijata chyba a výpočet syndromu nesouhlasí, je slovo po cyklickém posuvu na správný bit opraveno.

V následující tabulce jsou uvedeny dva případy dekódování a opravy chyb pro kód s generujícím mnohočlenem $g(x) = x^4 + x^3 + 1$. V prvním sloupci je zaznamenán krok opravy. Ve druhém až čtvrtém sloupci je uvedeno dekódování kódového slova bez chyby. V pátém až sedmém sloupci je znázorněna činnost dekodéru při výpočtu syndromu (v krocích 1-15) a opravy chyby (v korcích 16-30). [12] Funkce Meggitova dekodéru je zde uvedena na kódovém slově $v(x) = x^{10} + x^6 + x^2 + 1$. Chyba je reprezentována na pozici $e(x) = x^4$ a dekóduje se na základě syndromu $s(x) = x^3$ vypočítaného dělením přijatého slova $w(x) = x^{10} + x^6 + x^4 + x^2 + 1$ generujícím mnohočlenem.

| | bez chyby | | | chyba $e(x)=x^4$ | | |
|---|---|---|---|---|---|---|
| Krok | vstup | syndrom | oprava | vstup | syndrom | oprava |
| 1 | 0 | 0000 | | 0 | 0000 | |
| 2 | 0 | 0000 | | 0 | 0000 | |
| 3 | 0 | 0000 | | 0 | 0000 | |
| 4 | 0 | 0000 | | 0 | 0000 | |
| 5 | 1 | 1000 | | 1 | 1000 | |
| 6 | 0 | 0100 | | 0 | 0100 | |
| 7 | 0 | 0010 | | 0 | 0010 | |
| 8 | 0 | 0001 | | 0 | 0001 | |
| 9 | 1 | 0001 | | 1 | 0001 | |
| 10 | 0 | 1001 | | 0 | 1001 | |
| 11 | 0 | 1101 | | 1 | 0101 | |
| 12 | 0 | 1110 | | 0 | 1011 | |
| 13 | 1 | 0110 | | 1 | 0100 | |
| 14 | 0 | 0011 | | 0 | 0010 | |
| 15 | 1 | 0000 | | 1 | 1001 | |
| 16 | | 0000 | 0 | | 1101 | 0 |
| 17 | | 0000 | 0 | | 1111 | 0 |
| 18 | | 0000 | 0 | | 1110 | 0 |
| 19 | | 0000 | 0 | | 0111 | 0 |
| 20 | | 0000 | 1 | | 1010 | 1 |
| 21 | | 0000 | 0 | | 0101 | 0 |
| 22 | | 0000 | 0 | | 1011 | 0 |
| 23 | 0 | 0000 | 0 | 0 | 1100 | 0 |
| 24 | | 0000 | 1 | | 0110 | 1 |
| 25 | | 0000 | 0 | | 0011 | 0 |
| 26 | | 0000 | 0 | | 1000 | 1 |
| 27 | | 0000 | 0 | | 0100 | 0 |
| 28 | | 0000 | 1 | | 0010 | 1 |
| 29 | | 0000 | 0 | | 0001 | 0 |
| 30 | | 0000 | 1 | | 1001 | 1 |

Tab. č. 5 – Činnost Meggitova dekodéru

# 8 HRADLOVÁ POLE FPGA

FPGA neboli "Field programmable gate array" jsou integrované obvody navržené tak, aby mohly být konfigurovány zákazníkem, nebo programátorem pro různé typy úloh. Programování těchto polí se obecně zapisuje pomocí jazyků HDL (Hardware Description Language – jazyk pro popis hardware). Tyto jazyky zastávají tři funkce: popis systému, simulace systému a generace výrobních podkladů. Výrobní podklady jsou zde myšleny zejména jako vzájemné propojení prvků, jimiž disponuje cílová technologie.

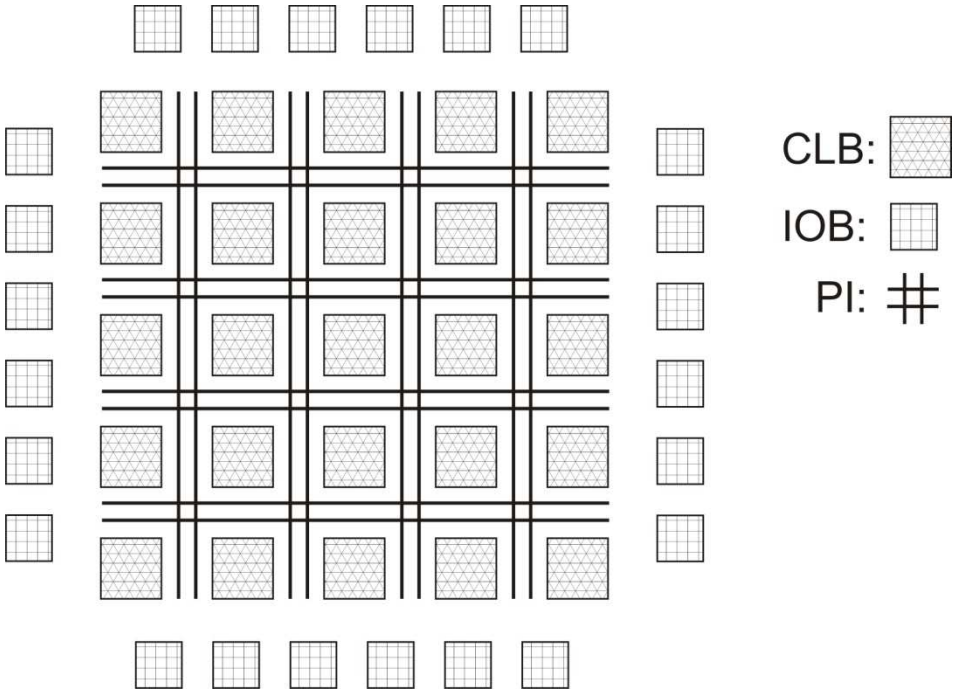
Tyto obvody se v současné době velice rychle rozvíjejí a dosahují rozsahu a výkonu, kterými v určitých oblastech konkurují signálovým procesorům, nebo je i v některých úlohách mohou předčít. Obvody FPGA nejsou speciálně zaměřeny jako obvody ASIC (Application Specific Integrated Circuit – integrovaný obvod přizpůsobený pro konkrétní použití), a tudíž mohou být použity při realizaci jakékoliv logické funkce, kterou realizovaly specializované ASIC s tou výhodou, že se nemusí vyrábět pro každý krok specializované obvody přímo na míru.

Obvody FPGA jsou nezastupitelnou součástí výroby tam, kde je potřeba měnit ve vyvíjených obvodech funkce a částečně rekonfigurovat jejich konstrukci bez toho, aby náklady na výrobu nevzrůstaly jako u ASIC obvodů. Kdybychom vyvíjeli ASIC pro každou vývojovou verzi, tak by se výroba výsledného obvodu velice prodražila.

FPGA obsahují programovatelné logické komponenty (logické bloky), a hierarchii re-konfigurovatelných propojení, které umožní jednotlivé bloky dávat dohromady. Tyto bloky poté mohou být propojeny k provádění složitých kombinačních funkcí, nebo jen jako jednoduchá logická hradla (AND, OR, XOR a další). Ve většině FPGA obvodů jsou také vytvářeny logickými bloky paměťové obvody, a to většinou jako flip-flop obvody.

Základní bloková struktura obvodů FPGA je naznačena na obrázku číslo osm níže. Je tvořena polem konfigurovatelných logických bloků (configurable logic block, CLB), které můžeme přirovnat k malým blokům obvodů CPLD. Bloky CLB se zpravidla ještě dělí na menší části nazývané jako logické buňky. Označení těchto prvků a jejich přesná struktura jsou odlišné u různých výrobců. Logické buňky ale většinou obsahují stejnou základní strukturu pro vytvoření kombinačních funkcí a podobné klopné obvody. Kombinační struktura je obvykle založena na principu struktury PROM s malým počtem vstupů – obvykle to bývají čtyři vstupy. Tento prvek se nazývá LUT (look-up table) a dovoluje vytvořit jen poměrně jednoduché funkce. Pro vytvoření složitějších logických

funkcí je nutno propojit více logických buněk, nebo jejich prvků LUT. K propojení bloků CLB slouží programovatelná propojovací struktura PI (programmable interconnect). Přičemž většina FPGA obvykle umožňuje propojit některé signály logických bloků přímo se sousedními bez nutnosti využívat globální propojovací matici. [13]




Obr. č. 9 – Základní bloková struktura obvodů FPGA

Jak je vidět z předchozího obrázku, pole bloků CLB je obklopeno vstupně-výstupními bloky (input/output block, IOB), které zajišťují podobné funkce jako vstupní a výstupní zesilovače. Obvykle také obsahují klopné obvody, nikoliv však kombinační logiku. Dále také mohou obsahovat registr, budič, multiplexer a ochranné obody. Jsou připojeny k vývodům obvodu FPGA a jejich úkolem je propojení vnějších signálů se signály v poli bloků CLB.

Kromě bloků znázorněných na předchozích obrázcích integrují výrobci do FPGA další prvky. U většiny moderních FPGA nechybí několik bloků rychlé synchronní statické paměti RAM. Také se integruje PLL (Phase Locked Loop) nebo DLL (Dealy Locked Loop), které se používají na obnovení charakteristik hodinového signálu, případně pro násobení nebo dělení jeho frekvence.

Úloha najít optimální propojení prvků struktury FPGA tak, aby výsledek plnil požadovanou funkci a přitom aby tato struktura byla efektivně využita, je však velice složité a její použití je dnes prakticky nemyslitelné bez použití podpory počítače, který musí být vybaven návrhovým systémem se složitými optimalizačními algoritmy. Dále mívají obvody FPGA vytvořenou strukturu pro použití většího množství hodinových signálů, speciální struktury pro generování rychlého přenosového signálu u sčítaček a další přídavné prvky.

Použití FPGA se uplatňuje například ve zpracování digitálního signálu, v leteckých systémech, navrhování ASIC obvodů, lékařských zobrazovacích systémech, dále se uplatňuje například při rozpoznávání řeči, kryptografii, bio-informatice, emulaci počítačového hardware v radioastronomii a mnoha dalších odvětvích. Také nachází uplatnění v oblastech, které plně využijí masivního paralelizmu jejich zapojení. Tato vlastnost je využita například pro kód lámání hesel, zejména při Brute-force útoku (útoku hrubou silou).

Z toho, co bylo uvedeno o struktuře obvodů FPGA, vyplývá, že s výjimkou nejjednodušších funkcí je každá funkce vytvořena v této struktuře svým vlastním specifickým propojením více (někdy také mnoha) logických buněk. Navíc se u obvodů FPGA uplatňuje zpoždění v propojovací struktuře. To má za následek, že analýza časových poměrů (například zpoždění signálu) je u obvodů FPGA mnohem složitější. Zatímco zpoždění v obvodech PLD lze u jednodušších aplikací odhadnout i bez použití počítačové podpory, u obvodů FPGA to je v dnešní době prakticky vyloučeno, a je nezbytné používat k tomu poměrně složité programové nástroje. [13]

## 8.1 Hlavní výrobci FPGA

V současné době jsou na trhu dva hlavní výrobci hradlových polí FPGA. Prvním a největším výrobcem je firma Xilinx, která sama ovládá více než 50 procent trhu s těmito hradlovými poli. Druhým největším výrobcem je firma Altera, která má 30 procent ze zbývající části tohoto trhu. [14] Ostatní konkurenční firmy jako jsou například SiliconBlue Technologies, Actel, Lattice Semiconductor, QuickLogic, Achronix si rozdělují zbylých 20 % trhu poskytujícího výrobu, servis a podporu hradlových polí FPGA.

Jedním z nejnovějších trendů je realizace 3D-FPGA, ve kterých jde především o to, že se do hradla přidá třetí rozměr a jednotlivé prvky se tak mohou propojovat do hustších a

bližších útvarů, což vede ke zrychlení celého obvodového prvku a realizaci funkcí s rychlostí asi o třetinu rychlejší, než je u normálních hradel FPGA. Tato hradla již byla realizována menšími firmami, například Tabula a TierLogic. Dvě největší firmy Xillinx a Altera zatím na toto hardwarové řešení FPGA obvodů nepřecházejí.

## 8.2 Altera jako výrobce hradlových polí FPGA

Jak již bylo řečeno, Altera Corporation je druhým největším výrobcem high-end PLD (programmable logic devices). Tato společnost je na trhu od roku 1983, kdy vydala své první programovatelné logické zařízení. Altera nabízí programovatelné obvody FPGA, CPLD a ASIC v kombinaci se softwarovými nástroji, které sami vyvíjí. Dále poskytuje širokou zákaznickou podporu více než 12 000 zákazníkům po celém světě. [15] Roční obrat této společnosti činil v roce 2009 v USA 1,2 miliardy dolarů. Společnost má centrálu v kalifornském San Jose a zaměstnává zhruba 2600 lidí v pobočkách 19 zemí světa.

V současné době firma Altera vyrábí a nabízí programovatelné hradla FPGA řady:

- Statix V,
- Arria II,
- Cyclone IV,
- Statix IV,
- Arria a
- Cyclone III

dále její sortiment zahrnuje CPLD s názvem MAX II. Jako jedinou řadu ASIC nabízí řadu s názvem HardCopy, kterou vyrábí od roku 2001. Poslední prototyp této řady s názvem “HardCopy V“ je vyráběn 28 nm technologií. Jednotlivé prvky jsou samozřejmě provázány: Po návrhu funkčního obvodu v hradlovém poli FPGA série “Stratix“ je možno tento návrh hladce převést do tištěné podoby a realizovat jej na ASIC obvodu „HardCopy V“ a začít tak sériovou výrobu.

Pro programátory a vývojáře má Altera velikou podporu, a to počínaje vývojovým prostředím Quartus II design software, které je od firmy Altera poskytováno zcela zdarma jen s menšími vývojovými omezeními. Dále je možnost si zažádat o takzvané duševní vlastnictví Altery (IP – Intelectual Property), kde je možno získat bloky různé složitosti a velikosti pro návrh, jako jsou bloky navržených paměťových řadičů, mikroprocesorů, bloky pro zpracování signálů, protokoly pro rozhraní a mnoho dalšího.

## 8.3 Vývojové prostředí

K vývoji aplikací pro FPGA je nutno pracovat s několika návrhovými systémy. Pro vytvoření aplikace je nutno nejméně dvou nástrojů. Prvním je nástroj pro syntézu, který převede většinou textový popis návrhu v některém HDL jazyce na netlist využívající obecné logické bloky. Druhý nástroj zajistí konverzi obecného netlistu na netlist využívající prostředky konkrétního FPGA a zajistí jejich optimální rozmístění a propojení.

Nástroje pro rozmístění a propojení obvykle nabízejí pouze výrobci programovatelných hradlových polí, ale prostředky pro syntézu mohou nabízet i jiné firmy. Pokud tedy chce vývojář začít pracovat na aplikaci, která bude simulována na FPGA, musí si obstarat základní programové vybavení od výrobce obvodů nebo jiný software od třetí strany.

Kromě nástrojů pro syntézu je velmi výhodné používat ještě simulátor, čímž se může předejít chybám již v průběhu návrhu. Ceny těchto vývojových prostředků jsou však velmi vysoké (ceny licencí na jeden rok se obvykle pohybují ve stovkách dolarů).

Některé z firem ovšem nabízí vývojové prostředí v neúplné verzi i zcela zdarma k obvodům s nižší hustotou logiky. Například vývojové prostředí do firmy Xilinx, které se jmenuje ISE WebPACK, je zcela zadarmo. Nebo vývojové prostředí od firmy Altera, které se jmenuje Quartus II web Edition, je také zdarma. Většina omezení se vztahuje na velikosti hradlových polí, které je možno v tomto prostředí použít. Xilinx ve svém prostředí navíc neuvádí plnohodnotný editor výsledného propojení. Většina návrhových systému poskytovaných zdarma obsahuje i nějakou verzi HDL simulátoru, které obsahují předkompilované knihovny funkcí pro FPGA. Například u firmy Xilinx se jmenuje simulátor „ModelSim XE“ a jeho omezení spočívá v tom, že je zpomalen pro velké návrhové funkce. Tudíž simulace složitějších funkcí mu trvá mnohonásobně déle. [16]

## 8.4 Popis vývojového prostředí Quartus II (Altera)

Pro realizaci návrhu jsem si vybral prostředí Quartus II design software. Toto prostředí je poskytováno firmou Altera ve dvou verzích a s podporou několika OS jak je vidět v následující tabulce:

| Software | Windows XP | | Windows Vista | | Red Hat Enterprise Linux 4/5 | | SUSE Linux Enterprise 9/10 | | CentOS 4/5 | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 32-bit | 64-bit | 32-bit | 64-bit | 32-bit | 64-bit | 32-bit | 64-bit | 32-bit | 64-bit |
| **Quartus II Subscription Edition** | Ano | Ano | Ano | Ano | Ano | Ano | Ano | Ano | Ano | Ano |
| **Quartus II Web Edition** | Ano | Ne | Ano | Ne | Ne | Ne | Ne | Ne | Ne | Ne |

Tabulka č. 6 – Podpora OS pro Quartus II

Jak je vidět z předchozí tabulky, první verze vývojového prostředí Quartus II s označením „Subscription Editon“ (předplacená edice) má podporu pro 32 i 64 bitové Windows XP, Vista (Seven) a také pro 32 a 64 bitové verze Linux (Red Hat Enterprise 4/5, SUSE Enterprise 9/10). Podporuje jeden z méně známých OS a to CentOS ver. 4/5. Tato verze se dá předplatit ve čtyřech variantách a to Subscription Renewal (licence pro jenoho na jeden rok za 2,495 USD), Fixed Node Subscription (licence pro jednoho na jeden rok s podporou aktualizací za 2,995 USD), Floating Node Subscription (toto roční předplatné podporuje veškeré Windows a Linux OS provázané v sítí a také poskytuje aktualizace za 3,995 USD) poslední variantou je Floating Node Additional Seat (přídavný účastník do sítě také za 3,995 USD).

Druhá verze vývojového prostředí Quartus II s označením „Web Edition“ podporuje jen 32 bitové Windows OS. Jelikož je Web Edition je tato vrze takto omezena je nabízena zcela zdarma a bez větších omezení. Quartus II Web Edition zahrnuje všechny potřebné prvky, které jsou potřeba pro vývoj aplikací na následujících Altera FPGA a CPLD těchto řad:

- Cyclone®, Cyclone II, Cyclone III, Cyclone IV and Arria® GX FPGAs
- All MAX® CPLDs
- Arria II GX FPGAs: EP2AGX45
- Stratix® III FPGAs: EP3SE50, EP3SL50, EP3SL70
- Stratix II and Stratix II GX FPGAs: EP2S15, EP2SGX30
- Stratix FPGA: EP1S10

Dále tato verze obsahuje: ModelSim Altera, což je začátečnická verze software od Mentor Graphics, která slouží pro VHDL nebo Verilog HDL simulace bez vyžadování licence. Tento balík také obsahuje vývoj pro vstupy, aplikaci syntézy a prostředky pro ověření návrhu. Dále také optimalizační nástroje pro vývoj.

# II. PRAKTICKÁ ČÁST

# 9 REALIZACE KODÉRU (12,8)

Podrobný popis algoritmu, kterým je kódováno informační slovo do systematického cyklického kódu, je uveden v bodě 6.2 – vytváření systematických cyklických kódů. Při realizaci obvodu tohoto algoritmu je pro dělení informačního mnohočlenu $u(x)$ generujícím mnohočlenem $g(x)$ použit posuvný registr se zpětnými vazbami. Tento posuvný registr po průchodu celého informačního mnohočlenu zobrazuje zbytek po dělení $r(x)$ generujícím polynomem $g(x)$ ve formě signálů na jednotlivých výstupech klopných obvodů.

Zbytek po dělení $r(x)$ je poté používán jako zabezpečovací část kódového slova a je připojen k informačním bitům na výstupu z kodéru. Spojením informačního mnohočlenu $u(x)$ a zbytku po dělení $r(x)$ vzniká kódové slovo $w(x)$, které na dekodéru zajistí rozpoznání dvou chyb a opravu jedné chyby v přijatém kódovém slově.

Zapojení bylo realizováno v prostředí „Quatus II version 9.1 web edition service pack 2“ od firmy Altera. Projekt byl vytvářen pro realizaci na hradle s označením „Cyclone II EP2C35F672C6“. Realizace kodéru a dekodéru na toto konkrétní hradlo zapříčinila použití specifických součástek právě pro tento konkrétní druh hradla. Proto jsou zde používané součástky uvedeny v konkrétním tvaru.

## 9.1 LFSR – obvod pro dělení polynomů

Základní částí obvodu pro kódování systematického cyklického Hammingova kódu (12,8)-kódu je obvod pro dělení informačního mnohočlenu $u(x)$ generujícím mnohočlenem $g(x)$. Tento obvod je realizován jako LFSR neboli „Linear Feedback Shift Register“. Obvod vykonává v prvních osmi krocích svojí činnosti dělení těchto dvou polynomů a po dokončení je z výstupů všech čtyř klopných obvodů odečten výsledný stav. Vyčtení leze realizovat paralelně i sériově. V tomto případě jsem zvolil vyčtení zbytku po dělení r(x) z obvodu sériově.

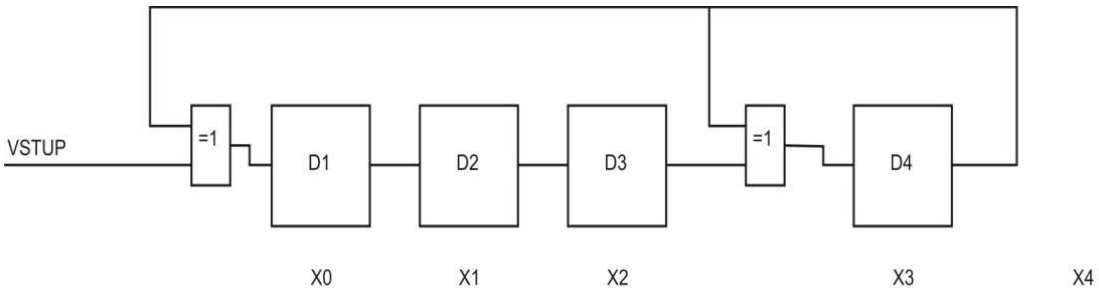
Jako generující mnohočlen jsem zvolil polynom $g(x) = x^4 + x^3 + 1$. Tento polynom jsem zvolil ze dvou možných generujících polynomů. První z nich je polynom, který byl mnou vybrán a to $g(x) = x^4 + x^3 + 1$, druhý generující polynom se uvádí ve tvaru $g(x) = x^4 + x + 1$ . Generující polynom musí splňovat následující pravidla: Musí beze zbytku dělit polynom $1 + x^n$ a také musí být stupně $n - k$.

### 9.1.1 Galois LFSR

Pro realizaci LFSR obvodu se používají dvě různé metody. První metoda se nazývá Fibonacciova a druhá metoda nese název Galoisova. Metoda Fibonacciona spočívá v tom, že jednotlivé klopené obvody jsou propojeny mezi sebou vzájemně, bez přerušení. Zpětná vazba prochází přes speciálně umístěné XOR členy zpět k prvnímu členu klopných obvodů. Toto zapojení jsem nerealizoval, protože mi přišlo složitější než Galoisova metoda.

Druhá metoda a zároveň metoda, kterou jsem použil, je metoda zapojení podle Galoise. Je pojmenována po francouzském matematikovi Evariste Galoisovi. Vyznačuje se tím, že mezi generující klopné obvody jsou zasazena hradla XOR, přes která prochází bit do dalšího klopného obvodu, a také je k nim připojena zpětná vazba z výstupu obvodu. Připojením této zpětné vazby přímo na členy XOR je ovlivněn výsledek na jednotlivých klopných obvodech téměř okamžitě.

Realizace obvodu pro generující mnohočlen $g(x) = x^4 + x^3 + 1$ s výstupem čtyř zabezpečovacích bitů (zde uvedena jako zbytek po dělení) je provedena čtyřmi klopnými obvody (označené jako $D_1$, … $D_4$) a dvěma hradly XOR. Toto zapojení je znázorněno na následujícím blokovém schématu.

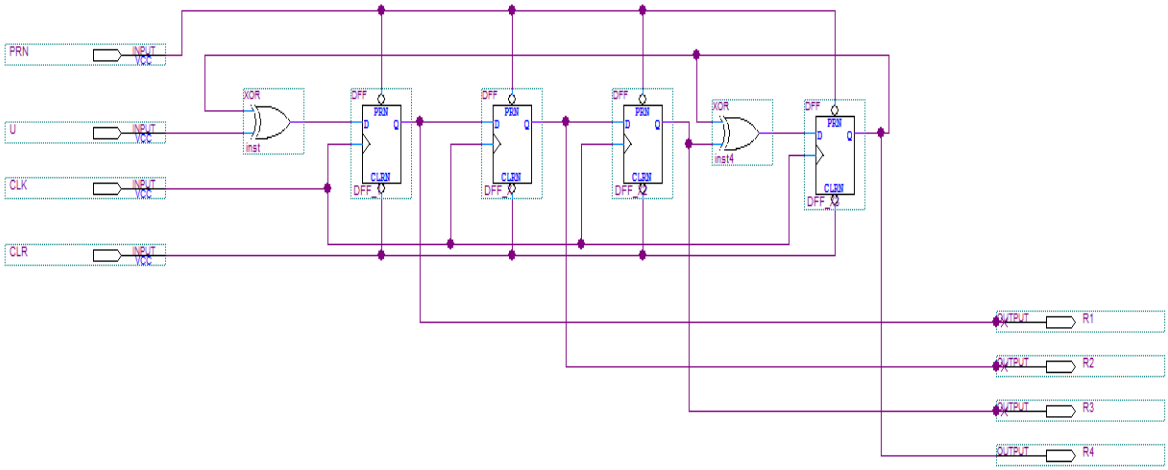



Obr. č. 10 – Blokové schéma LSFR

Z předchozího obrázku vyplývá, že zapojení hradla XOR je vždy před klopný obvod, kterým je reprezentován bit na pozici generujícího polynomu. Tudíž hradlo XOR musí být v mém případě na pozici před prvním klopným obvodem a před třetím. A po průchodu všech osmi bitů bude stav klopných obvodů odpovídat zbytku po dělení za předpokladu, že použiji osm informačních bitů. Pokud bych tento obvod použil pro jedenácti bitové slovo, zbytek po dělení r(x), by byl na klopných obvodech až po příchodu posledního (jedenáctého) bitu.

## 9.2 Zapojení obvodu pro dělení mnohočlenů v prostředí Quartus II

Předchozí zapojení obvodu pro dělení mnohočlenů jsem provedl v prostředí Quartus II pomocí „Block diagram file“ (součásti vývojového prostředí na vytváření obvodů pomocí blokových diagramů). Toto zapojení je znázorněno na obrázku č. 11, ve kterém jsou vidět použité součástky.




Obr. č. 11 – Zapojení LSFR v prostředí Quartus II

Ve schématu je vidět, že do obvodu vedou čtyři vstupy (PRN, U, CLK, CLR). Všechny tyto vstupy jsou realizované jako „std_logic“ (vstupy s logickou hodnotou). Vstup „U“ představuje vstup informačního slova a vstupem CLK jsou realizovány hodiny. Použité klopné obvody pracují na základě následující tabulky.

| **Vstupy** | | | | **Výstup** |
|---|---|---|---|---|
| **PRN** | **CLRN** | **CLK** | **D** | **Q** |
| L | H | X | X | H |
| H | L | X | X | L |
| L | L | X | X | L |
| H | H | nab. hrana | L | L |
| H | H | nab. hrana | H | H |
| H | H | L | X | Qo |
| H | H | H | X | Qo |

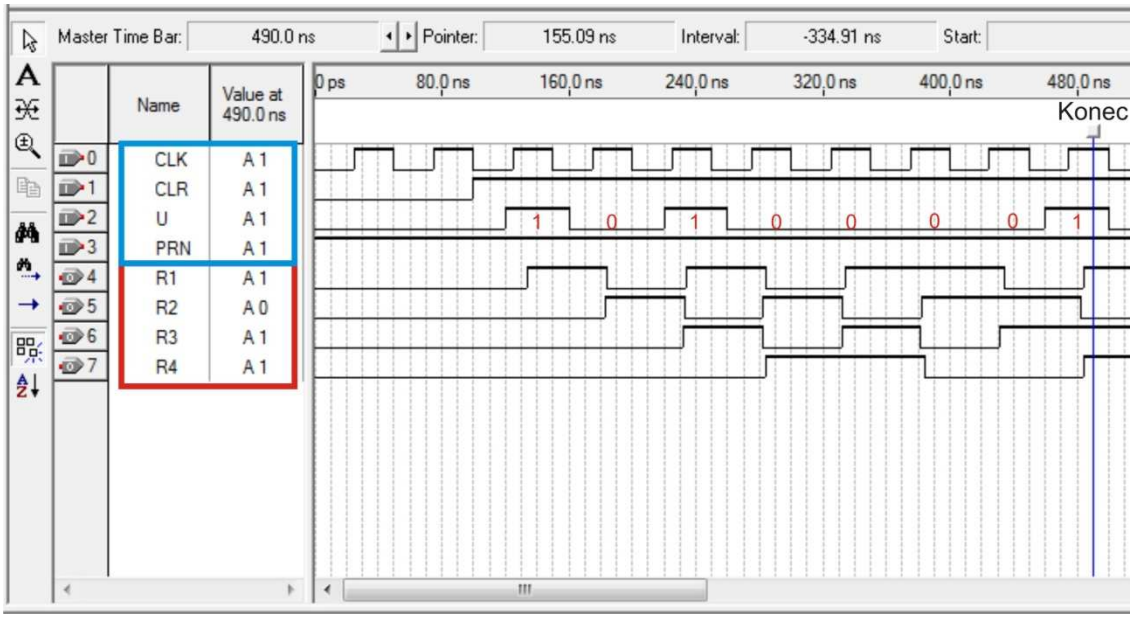
Tab. č. 7 – Tabulka pro použitý klopný obvod

Čtyři výstupy (R1, … R2) vyvedené z obvodu jsou průběžné stavy klopných obvodů v průběhu průchodu informačního slova. Po průchodu všech osmi bitů informačního slova je na výstupu zbytek po dělení informačního mnohočlenu $u(x)$ generujícím členem $g(x)$. Tyto výstupy jsou vyvedeny jen pro kontrolu správnosti výpočtu a z výsledného obvodu pro kódování a dekódování budou vynechány.

## 9.3 Ověření funkčnosti LSFR obvodu

Obvodové zapojení v prostředí Quartus II jsem ověřil na dvou praktických příkladech průchodu informačního slova tímto obvodem a následným odečtením stavů klopných obvodů.

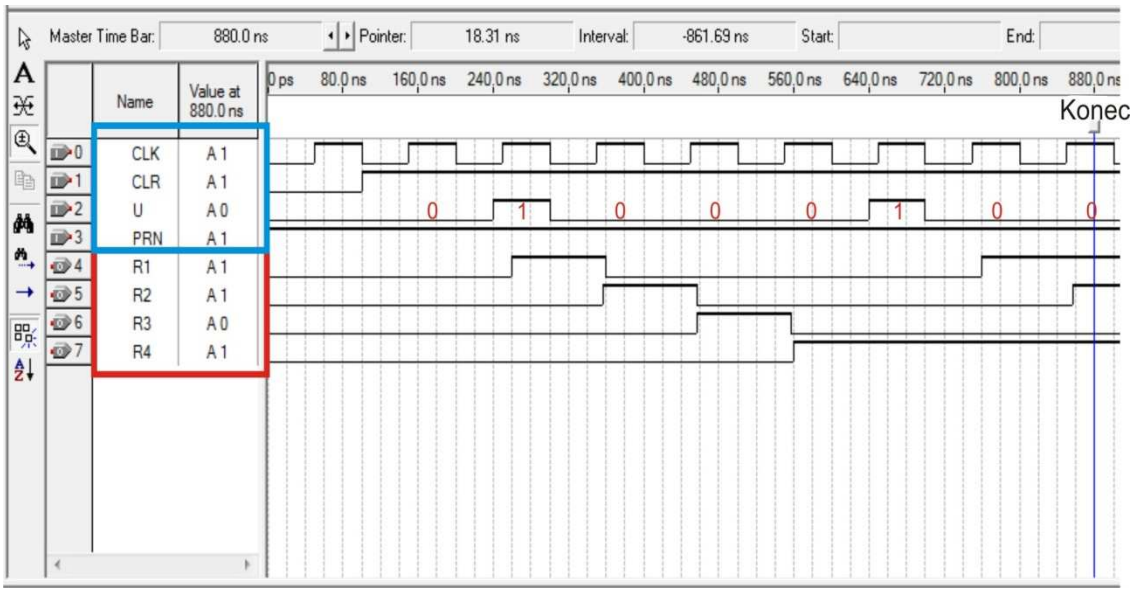
V prvním testu jsem použil osmibitové informační slovo ve tvaru $u(x) = x^7 + x^5 + 1$ (v binární podobě: 10100001). Po vydělení tohoto mnohočlenu generujícím mnohočlenem $g(x) = x^4 + x^3 + 1$ jsem dostal čtyřbitový zbytek po dělení $r(x) = x^3 + x^2 + 1$ (v binární podobě: 1101). Ověření při průchodu obvodem, jak je vidět z následujícího obrázku, dopadlo stejně.




Obr č. 12 – Ověřesní LSFR obvodu na náhodném informačním slově test 1

Modrým obdélníkem jsou označeny čtyři vstupy do obvodu LSFR a červeným obdélníkem jsou označeny čtyři bitové výstupy vycházející z obvodu. Na obrázku je svislou modrou čárou označen konec informačních bitů a zároveň je v této chvíli odečtena hodnota na jednotlivých hradlech R1, …. , R4, která dává výsledek shodný se zbytkem po dělení polynomu $u(x)$ polynomem $g(x)$ a to 1101.

V druhém testu jsem použil taktéž osmibitové informační slovo ve tvaru $u(x) = x^6 + x^2$ (v binární podobě: 01000100). Po vydělení tohoto mnohočlenu generujícím mnohočlenem $g(x) = x^4 + x^3 + 1$ jsem dostal čtyřbitový zbytek po dělení $r(x) = x^3 + x + 1$ (v binární podobě: 1011). Ověření při průchodu obvodem, jak je vidět z následujícího obrázku, dopadlo stejně.




Obr č. 13 – Ověřesní LSFR obvodu na náhodném informačním slově test 2

Obrázek je označen stejně jako předchozí simulační obrázek. Svislou modrou čárou je vyznačen konec informačních bitů a zároveň je v této chvíli odečtena hodnota na jednotlivých hradlech R1, …. , R4, která dává výsledek shodný se zbytkem po dělení polynomu $u(x)$ $polynomem$ $g(x)$.

## 9.4 Vytvoření kódového slova

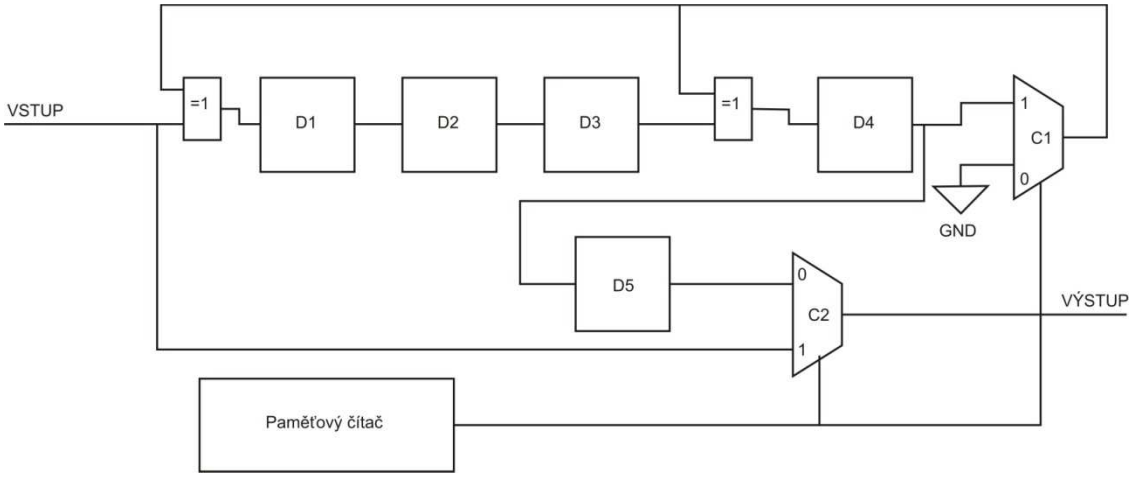
Pro vytvoření kódového slova je třeba k obvodu pro dělení mnohočlenů připojit ještě dva multiplexory a jeden adresový čítač, aby obvod vytvářel celé kódové slovo.

Prvních z multiplexorů pro zvolený kód (12,8) propustí v prvních osmi krocích (krok se bere jako jedna náběžná hrana hodin) informační slovo na výstup a poté adresový čítač přepne na druhý multiplexor, který do obvodu pro dělení mnohočlenů začne posílat samé nuly a tím vyřadí z provozu součástky XOR. Vyřazením součástek XOR z provozu vzniká z obvodu pro dělení jen posuvný registr a je možné z něj zbytek po dělení vyčíst sériově na

výstupu posledního klopného obvodu. Odpadá tak řešení paralelního odečítání hodnot a následné sériové posílání těchto hodnot na výstup.

## 9.5 Zapojení multiplexorů do obvodu

Z následujícího blokového diagramu je vidět, že první z multiplexorů s označením C1 je do obvodu zapojen tak, aby propouštěl v obvodu na dělení polynomů hodnoty, které mu přijdou na vstup. Druhý multiplexor s označením C2 propouští v prvním stavu (osm náběžných hran hodin) vstupní slovo u(x). Po průchodu určitého počtu bitů (v tomto případě osmi informačních) adresový čítač, který je také připojen na hodinové impulsy, provede změnu stavu multiplexorů a multiplexor s označením C1 začne do obvodu pro dělení vysílat samé nuly. Přivedení nul bylo docíleno tak, že druhý vstup multiplexoru C1 byl připojen na zem, která dává logickou nulu stále. Tyto nuly zapříčiní uchování stavů na jednotlivých D klopných obvodech (D1, … , D4) a jejich posunutí vpravo vždy o jeden krok za jeden hodinový impuls. Cokoliv je přivedeno na vstup, tak podle tabulky XOR nemůže změnit stav klopných obvodů a tudíž ovlivnit sériový výstup na klopném obvodu D4. Lze tedy stav vyčíst sériově po jednotlivých krocích. Přidáním klopného obvodu D5, který jsem umístil za obvod D4, bylo docíleno u tohoto zapojení zpoždění o jeden krok, který je potřeba ke správnému připojení zabezpečovací části r(x) k informační části u(x). Přidání klopného obvodu D5 je vidět na obrázku č. 13.




Obr. č. 14 – Umístění multiplexorů v obvodu kodéru

| X | Y | O |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Tab. č. 8 - XOR

### 9.5.1 Multiplexor

Pro realizování multiplexoru bylo v prostředí Quartus II využito prostředí pro HDL návrhy, ve kterém byla součástka napsána jazykem VHDL. Poté byla převedena na symbolu pro využití v Blokovém schematickém návrhu, ve kterém byl kodér realizován. VHDL kód je uveden v následujícím odstavci:

```
library ieee;
USE ieee.std_logic_1164.all;

entity mux_2_1 is
port (I0, I1, a: in std_logic;
  y: out std_logic);
  end mux_2_1;

architecture muxJIR of mux_2_1 is
  begin
          process (a,I0,I1) begin
                   if a = '1' then y <= I0;
                  else y <= I1;
                  end if;
          end process;
  end muxJIR;
```

### 9.5.2 Adresový čítač

Pro realizování adresového čítače jsem v prostředí Quartus II taktéž využil komponentu pro HDL návrhy, ve kterém jsem součástku napsal jazykem VHDL. Poté byla převedena na symbolu pro využití v Blokovém schematickém návrhu, ve kterém jsem celý kodér realizoval. VHDL kód adresového čítače je uveden v následujícím odstavci:

```
library ieee;
USE ieee.std_logic_1164.all;

entity counter is
port (cl, res : in std_logic;
out_a : out std_logic);
end counter;

architecture count_8_12 of counter is
signal a: bit_vector (0 to 3);
signal as: bit_vector (0 to 3);
        begin
                process (cl,res)
                        begin
                                if (res = '0') then
                                        a <= "0000";
                                end if;
                                if (cl = '0') and (res = '1') then
                                case a is
                                        when "0000" => as <= "0001"; a <= "0001";
                                                out_a <= '0';
                                                when "0001" => as <= "0010"; a <= "0010";
                                                when "0010" => as <= "0011"; a <= "0011";
                                                when "0011" => as <= "0100"; a <= "0100";
                                                when "0100" => as <= "0101"; a <= "0101";
                                                when "0101" => as <= "0110"; a <= "0110";
                                                when "0110" => as <= "0111"; a <= "0111";
                                                when "0111" => as <= "1000"; a <= "1000";
                                                when "1000" => as <= "1001"; a <= "1001";
                                                out_a <= '1';
                                                when "1001" => as <= "1010"; a <= "1010";
                                                when "1010" => as <= "1011"; a <= "1011";
                                                when "1011" => as <= "1100"; a <= "1100";
                                                when others => as <= "0000"; a <= "0000";
                                                out_a <= '0';
                                        end case;
                                end if;
                        end process;
                end count_8_12;
```

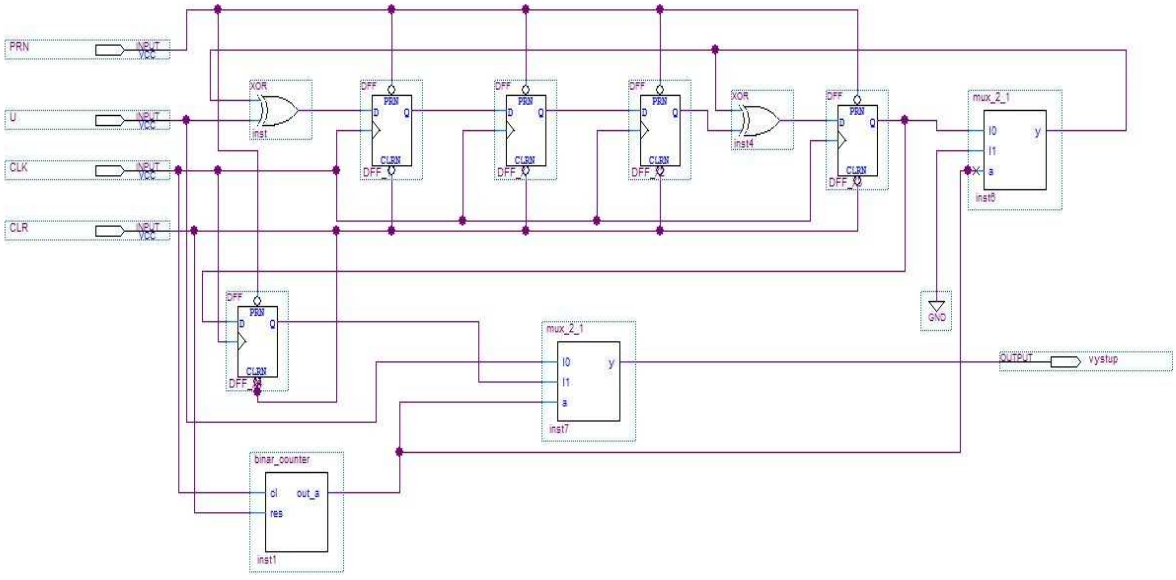
## 9.6 Ověření funkce kodéru

Zapojení kodéru (12,8) bylo provedeno s pomocí blokového diagramu v prostředí Quartus II. Na následujícím obrázku je znázorněn kodér se čtyřmi vstupy, které jsou potřeba pro správný chod součástek na tomto hradle. Popis jednotlivých vstupů:

| Vstup | Funkce |
|---|---|
| PRN | log. 1 |
| U | vstupní slovo |
| CLK | Hodinové impulzy |
| CLR | log. 1 |

Tab. č. 9 – Popis vstupů Modulátoru

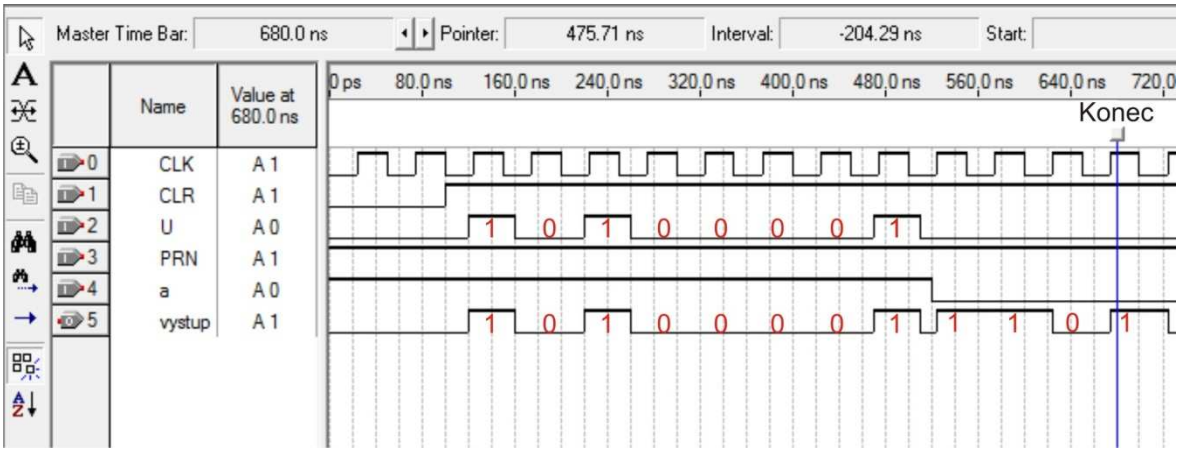
Po přivedení logické jedničky na vstupy PRN a CLR je možné za použití hodinových impulzů ovládat klopné obvody podle tabulky č. 7, která byla uvedena v bodu 9.2. Adresový čítač přepíná výstupy multiplexorů. Jak lze vyčíst z předchozího VHDL kódu adresového čítače. V prvních osmi bitech vysílá na multiplexory logickou nulu a poté ve čtyřech krocích vysílá na vstupy multiplexorů logickou jedničku. Tím dochází k přepínání multiplexorů.




Obr. č. 15 – Zapojení kodéru pro kód (12,8)

Obvodové zapojení v prostředí Quartus II jsem ověřil na dvou praktických příkladech průchodu informačního slova tímto obvodem a následným odečtením stavu na jediném výstupním pinu. Pro simulaci funkčnosti jsem vytvořil Vector Waveform file se čtyřmi vstupy „PRN, U, CLK, CLR“, které fungují podle tabulky č. 7, a jedním výstupem, který je znázorněn na předchozím obrázku úplně vpravo.

V prvním testu jsem použil informační osmibitové slovo ve tvaru $u(x) = x^7 + x^5 + 1$ (v binární podobě: 10100001). Po vydělení tohoto mnohočlenu generujícím mnohočlenem $g(x) = x^4 + x^3 + 1$ jsem dostal zbytek po dělení $r(x) = x^3 + x^2 + 1$ (v binární podobě: 1101). Následně jsou tato dvě slova vyslána za sebou na výstup, jak je vidět na obrázku č. 14. Dále jsou zde vidět ještě signály vstupů CLR, PRN. Signál „a“ je signál vycházející z adresového čítače, který přepíná multiplexory. Simulace byla uskutečněna jako funkční simulace bez zohledňování časového průchodu signálu součástkami. Proto zde není možnost pozorování stavu, kdy se jednotlivé klopné obvody překlápějí a nevzniká zde žádné zpoždění. Na obrázku je jasně vidět, že při každé náběžné hraně se na výstup dostává postupně celé kódové slovo $w(x) = (x^{11} + x^9 + x^4 + x^3 + x^2 + 1)$ v binární podobě „101000011101“. Toto slovo je tedy složeno z informační části „10100001“ a zabezpečovací části „1101“.

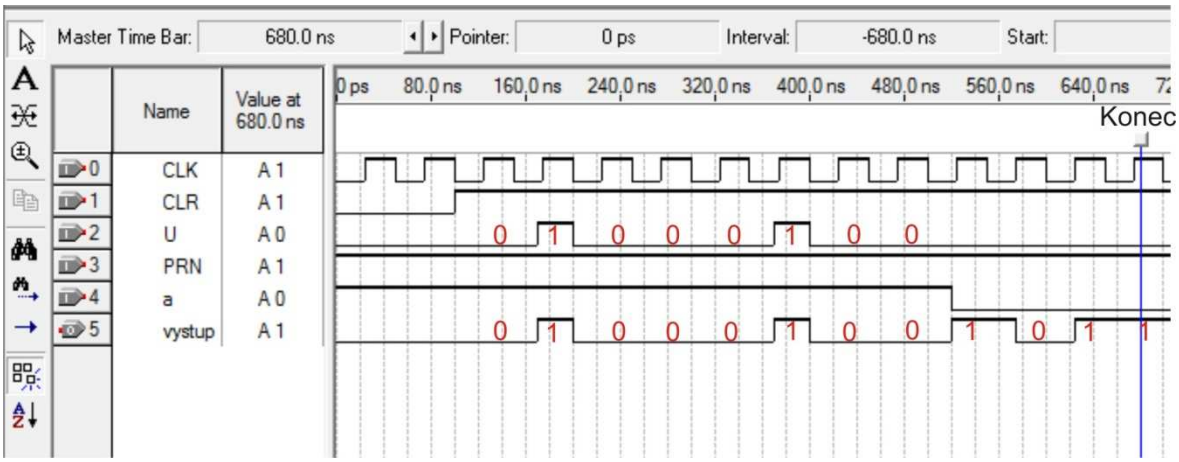



Obr. č. 16 - Kódované slovo 101000011101

Jak je vidět z obrázku, informační slovo u(x) zde uvedené jako vstup U, které přišlo na vstup kodéru je odesláno z výstupního pinu v nezměněné podobě, ale s přidanými čtyřmi zabezpečovacími bity r(x). Výstupní slovo znázorňuje druhá červená (výstupní) řada bitů.

V druhém testu jsem použil osmibitové informační slovo ve tvaru $u(x) = x^6 + x^2$ (v binární podobě: 01000100). Po vydělení tohoto mnohočlenu generujícím mnohočlenem $g(x) = x^4 + x^3 + 1$ jsem dostal zbytek po dělení $r(x) = x^3 + x + 1$ (v binární podobě: 1011). Následně jsou tato dvě slova vyslána za sebou na výstup, jak je vidět z následujícího obrázku. Dále jsou zde vidět stejné signály jako v předcházejícím případě: CLR, PRN a „a“. Tato simulace byla také uskutečněna jako funkční simulace bez zohledňování časového průchodu signálu součástkami.

Na obrázku je jasně vidět, že při každé náběžné hraně se na výstup dostává postupně celé kódové slovo ve tvaru $w(x) = (x^{10} + x^6 + x^3 + x + 1)$ v binární podobě „010001001011“. Toto slovo je tedy složeno z informační části „01000100“ a zabezpečovací části „1011“.




Obr. č. 17 – Kódované slovo 010001001011

Jak je vidět z obrázku, informační slovo u(x), zde uvedené jako vstup U, které přišlo na vstup kodéru je odesláno z výstupního pinu v nezměněné podobě, ale s přidanými čtyřmi zabezpečovacími bity r(x). Výstupní slovo znázorňuje druhá červená výstupní signálová řada bitů. Zabezpečovací čtyřbitová část je odečtena z LFSR obvodu, jak bylo popsáno výše.

Celý kodér je převeden na součástky pro realizaci na hradle Cyclone II obvodem, který je znázorněn v příloze č. 1 jako RTL obvod. Je zde vidět, že při implementaci obvodu na hradlové pole Cyclone II jsou tři klopné obvody sloučeny do jednoho a dva klopné obvody zůstaly nezměněny. Stejně tak zůstaly nezměněny i oba dva multiplexory. (viz. příloha č. 1)

Ze simulací vyplývá, že návrh a realizace systematického cyklického kodéru dopadla podle očekávání dobře. Jednotlivé součástky zapojené do tohoto speciálního obvodu spolu vzájemně fungovaly a dokázaly tak zakódovat příchozí slovo u(x) na informační slovo w(x).

# 10 REALIZACE MEGGITOVA DEKODÉRU (12,8)

Nejprve popíšu princip sestavení dekodéru a poté uvedu dva jednoduché příklady pro ověření funkce. Konstrukce dekodéru byla realizována pomocí Meggitova dekodéru jehož vlastnosti jsou založeny na vlastnostech cyklických kódů. Pokud nastane chyba v přijatém slově w(x), oprava tohoto slova se bude provádět a vypočítaný syndrom bude nenulový. Syndrom se zjistí tak, že provedeme dělení přijatého slova w(x) generujícím polynomem g(x). Pokud proběhne dělení bezezbytku, to znamená s nulovým syndromem, je přijaté slovo ve správném tvaru a není jej potřeba opravovat.

Pokud při dělení vznikne zbytek, znamená to, že při přenosu nastala chyba a toto slovo je potřeba opravit. Opravu je možné provádět, jen když slovo přišlo s jedním změněným bitem. U dvou bitů je oprava již nemožná. Pro opravu libovolného bitu musíme znát všechny syndromy, které mohou vzniknout chybou v každém bitu. Syndromy však nemusíme uchovávat v paměti, nýbrž si tyto syndromy můžeme průběžně vypočítat. K tomuto účelu jsem využil již dříve mnou vytvořený obvod pro dělení mnohočlenů LFSR (kapitola 9.2).

## 10.1 Dělení přijatého polynomu w(x)

Při dělení polynomů v obvodu LSFR dochází k vytváření tabulky syndromů pro každé přijaté slovo. Každý syndrom vyjadřuje chybu v určitém bitu přijatého kódového slova. Tento výpočet uvedu na příkladech dvou přijatých slov.

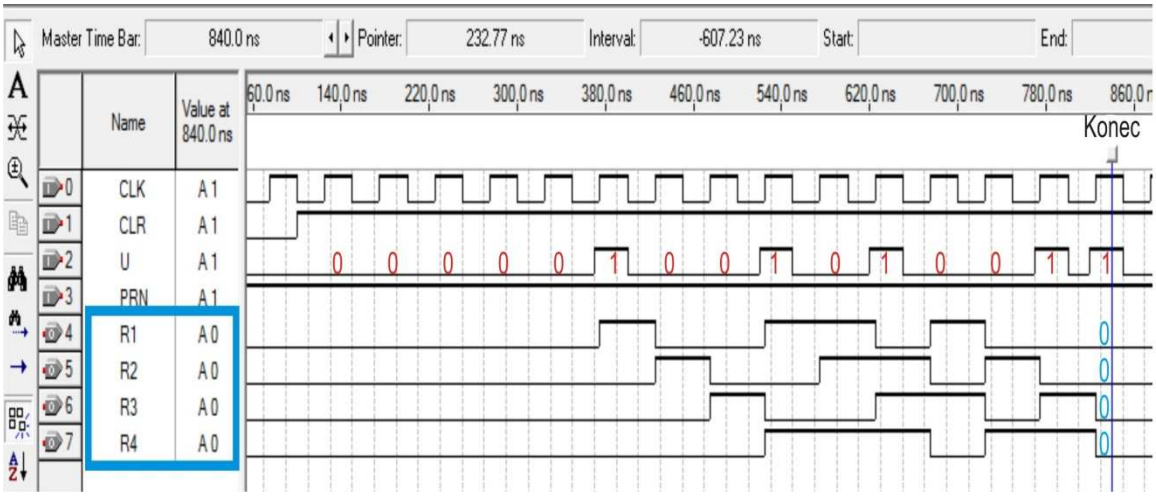
První přijaté slovo má tvar $w(x) = x^9 + x^6 + x^4 + x + 1$ a v binární podobě má tvar 000001001010011, to znamená patnáct bitů (využiji stejně jen osm informačních a čtyři zabezpečovací). Dělení probíhá v obvodu LSFR, který je znázorněn na obrázku číslo deset.

Toto slovo je přijaté správně a tudíž by měl být syndrom po vydělení generujícím polynomem $g(x) = x^4 + x^3 + 1$ roven nule. Při ověření výpočtem se potvrdilo, že zbytek po dělení polynomu w(x) polynomem g(x) opravdu odpovídá předpokládané nulové hodnotě.

$$(x^9 + x^6 + x^4 + x + 1) : (x^4 + x^3 + 1) = x^5 + x^4 + x^3 + x + 1$$

$$r(x) = 0$$

Tento výpočet jsem ověřil v obvodu na dělení polynomů LFSR polynomem $g(x) = x^4 + x^3 + 1$ z obrázku deset. Jak je vidět na následujícím „Waveform simulaci“, syndrom po vydělení je opravdu nulový a můžeme tak tedy tvrdit, že přijaté slovo w(x), jsme obdrželi v pořádku.




Obr. č. 18 – Dělení správně přeneseného slova w(x)

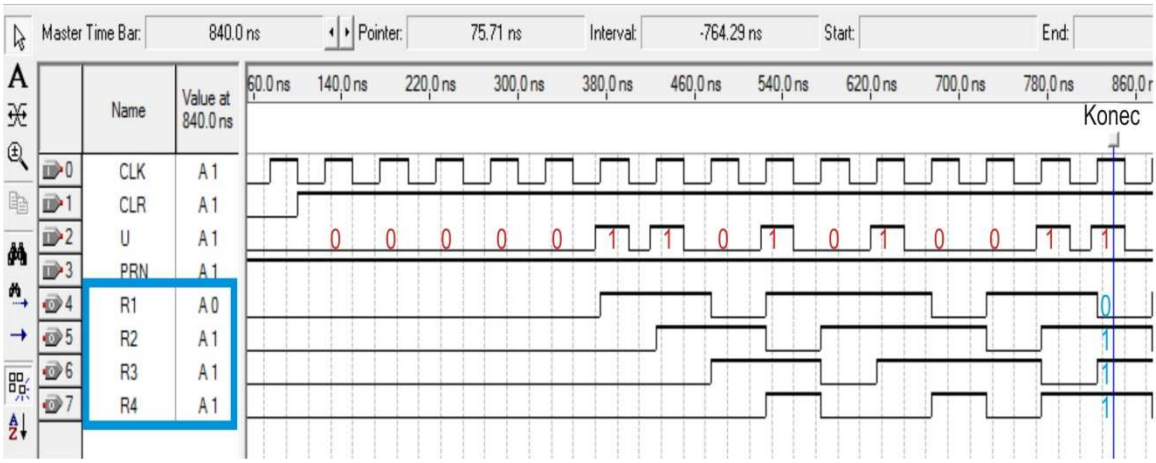
Z obrázku je vidět, že po průchodu celého slova děličkou LSFR je na jednotlivých výstupech klopných obvodů nula. Toto značí správně přijaté slovo. Červeně jsou vyznačeny bity přijatého slova w(x) a modrou barvou je vyznačen výsledek po dělení generujícím polynomem g(x).

V druhém případu jsem zvolil přijaté slovo w(x) stejné jako první přijaté slovo, akorát s tím rozdílem, že jsem udělal na pozici bitu $x^8$ chybu. Slovo tedy bylo ve tvaru $w(x) = x^9 + x^6 + x^4 + x + 1$ a v binární podobě má tvar 000001101010011. Po vydělení slova generujícím polynomem g(x) jsem dostal nenulový zbytek po dělení, který značí špatně přijaté kódové slovo.

$$(x^9 + x^8 + x^6 + x^4 + x + 1) : (x^4 + x^3 + 1) = x^5 + x^2 + 1$$

$$r(x) = x^3 + x^2 + x$$

Tento výpočet jsem ověřil v obvodu pro dělení LFSR polynomem $g(x) = x^4 + x^3 + 1$ z obrázku deset. Jak je vidět na následujícím obrázku (č. 18), syndrom po vydělení není nulový. Po vydělení zůstává na klopných obvodech polynom $r(x) = x^3 + x^2 + x$.




Obr. č. 19 – Dělení nesprávně přeneseného slova w(x)

Z obrázku je patrné, že po průchodu celého slova děličkou LSFR je na jednotlivých výstupech klopných obvodů hodnota „0111“. Červeně jsou vyznačeny bity přijatého slova w(x) a modrou barvou je vyznačen výsledek po dělení generujícím polynomem g(x). Nenulové hodnoty syndromu na klopných obvodech značí nesprávně přijaté slovo. Z následující tabulky lze vyčíst, že chyba podle syndromu $r(x) = x^3 + x^2 + x$ nastala na pozici $x^8$.

Tabulka pro zjištění chyby v určitém bitu je odvozena od syndromů, které zůstávají na jednotlivých klopných obvodech obvodu pro dělení polynomů (LSFR), po vydělení příchozího slova w(x) generujícím polynomem g(x). Platí zde princip superpozice kódového slova a slova s jednou jedničkou. Pokud tato jednička přijde na vstup jako první, okamžitě spustí výpočet syndromu všech patnácti syndromů. Tabulka obsahuje syndromy patnácti možných stavů, protože jak bylo řečeno v teorii, dekodér (12,8)-kódu je realizovaný z dekodéru (15,11)-kódu vynecháním slov s nejvyššími bity. Červeně označené syndromy nebudou využity.

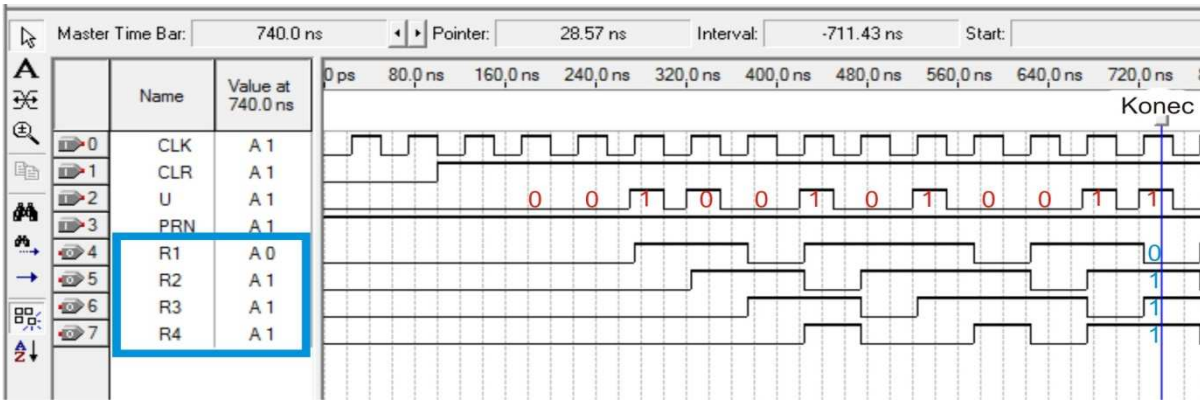
| Syndrom | Poloha chyby | Syndrom | Poloha chyby | Syndrom | Poloha chyby |
|---|---|---|---|---|---|
| 0 | bez chyby | $X^3+X+1$ | $X^5$ | $X^3+X^2+1$ | $X^{11}$ |
| 1 | 1 | $X^3+X^2+X+1$ | $X^6$ | $X+1$ | $X^{12}$ |
| X | X | $X^2+X+1$ | $X^7$ | $X^2+X$ | $X^{13}$ |
| $X^2$ | $X^2$ | $X^3+X^2+X$ | $X^8$ | $X^3+X^2$ | $X^{14}$ |
| $X^3$ | $X^3$ | $X^2+1$ | $X^9$ | | |
| $X^3+1$ | $X^4$ | $X^3+X$ | $X^{10}$ | | |

Tab. č. 10 – Tabulka syndromů chyb pro jednotlivé bity

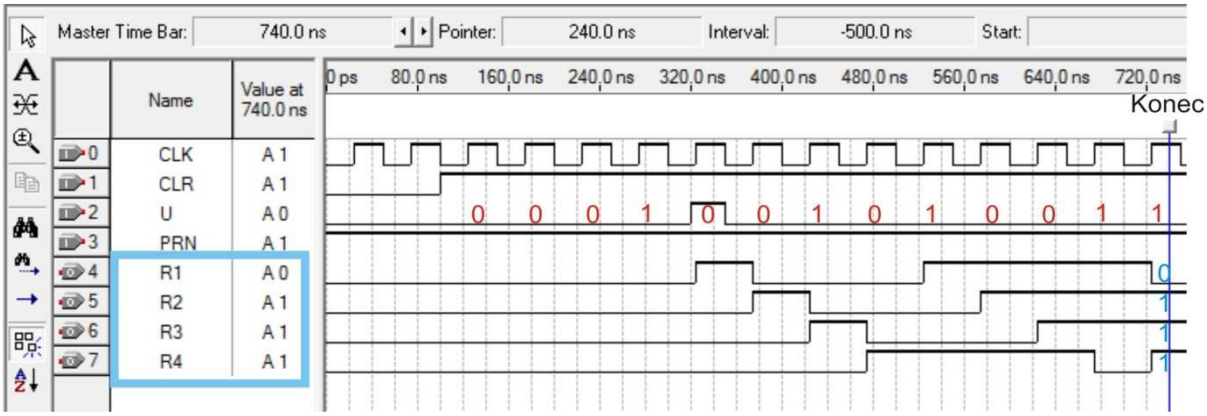
## 10.2 Obvod Meggitova dekodéru

V předchozí kapitole jsem uvedl, jak se vypočítává syndrom pro opravu chyby v systematickém cyklickém kódu (12,8)-kódu respektive (15,11)-kódu. Pro obvod Memggitova dekodéru je potřeba tento syndrom umět vhodně využít. V roce 1960 přišel Meggit na skutečnost, že syndrom přijatého slova s chybou a syndrom samostatného chybového slova mají ekvivalentní zbytek, jsou-li děleny stejným generujícím polynomem. V mém případě je to polynom $g(x) = x^4 + x^3 + 1$, který jsem využil i pro kódování.

Příklad výpočtu syndromu pro přijaté slovo s chybou a pro přijaté chybové slovo jsou uvedeny v následujících dvou obrázcích. Na prvním z nich (č. 18) vidíme, že přijaté slovo $w(x) = x^9 + x^6 + x^4 + x + 1$ a (v binární podobě má tvar 001101010011) má přijatý syndrom $r(x) = x^3 + x^2 + x$ (v binární podobě 1110). V dalším obrázku (č. 19) vidíme, že syndrom po průchodu stejného počtu bitů s chybou je stejný.



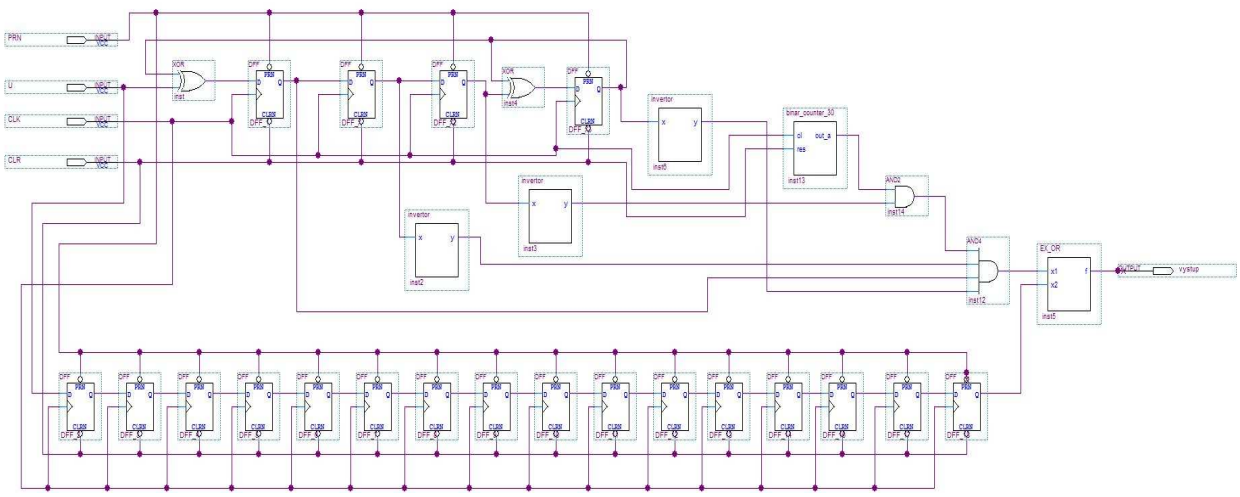

Obr. č. 20 – Výpočet syndromu pro špatně přijaté slovo




Obr. č. 21 – Výpočet syndromu pro přijatou chybu

Z obou předchozích obrázků je patrné, že syndrom přijatého slova s chybou je stejný jako syndrom přijaté chyby. Právě této vlastnosti je využito při realizaci obvodu Meggitova dekodéru.

Pro opravu chyby jsem použil syndrom v binární podobě „1000“. Když opravuji přijaté slovo w(x), musím jej při použití tohoto syndromu o patnáct kroků zpozdit, což jsem realizoval posuvným registrem. Tento registr jsem vytvořil ze šestnácti klopných obvodů, jak je vidět na následujícím obrázku. Pro opravu příslušeného bitu slouží obvod, do kterého jsem umístil tři invertory a dvě hradla AND. Dále je zde umístěn čítač, který zamezuje hradlu AND posílat opravné bity před příchodem slova z posuvného registru. Výstup hradla AND při syndromu 1000 nabývá hodnoty „1“. Tato hodnota pak provede opravu příslušného bitu pomocí hradla EX-OR, které je na obrázku nejvíce vpravo.



Obr. č. 22 – Meggitův dekodér

Z předchozího obrázku je vidět, že tři invertory umístěné za druhým, třetím a čtvrtým klopným obvodem posílají jedničky na hradlo AND, právě když je syndrom 1000. Na hradle AND se potom tyto signály sčítají. Za čtvrtý klopný obvod jsem umístil ještě jedno dvouvstupé hradlo AND pro omezení výstupu opravného syndromu před patnáctým krokem. Čítač před dvouvstupovým hradlem AND vysílá prvních 15 kroků hodnotu „0“ a poté dalších 15 kroků hodnotu „1“. Pokud je na všech vstupech obou hradel jednička, je poté vyslána do hradla EX-OR a zde proběhne oprava příslušného bitu ve zpožděném slově. Pro opravu patnáctibitového slova je potřeba 30 kroků dekodéru (náběžných hran hodin) a tento počet není možné zkrátit z důvodu správné funkce dekodéru.

## 10.3 Vytvoření součástek

Pro realizování jednotlivých součástek v prostředí Quartus II jsem využil prostředí pro HDL návrhy, ve kterém jsem součástky popsal jazykem VHDL. Tyto součástky byly poté převedeny na symboly, aby je bylo možno použít v Blokovém schematickém návrhu, ve kterém byl dekodér realizován.

### 10.3.1 VHDL návrh invertoru

```
library ieee;
USE ieee.std_logic_1164.all;

entity invertor is
        port ( x: in std_logic;
                y: out std_logic);
        end invertor;

Architecture inv of invertor is
begin
        process (x) begin
                 if x = '1' then y <= '0';
                else y <= '1';
                end if;
        end process;
end inv;
```

### 10.3.2 VHDL návrh čítače

VHDL návrh čítače je uveden v příloze č. 3 z důvodu velikosti jeho kódu. Tento čítač byl realizován jako pětibitový a jak už bylo řečeno dříve, tak v prvních patnácti krocích má na výstupu nulu a dalších patnáct kroků jedničku. Tato vlastnost zamezuje opravným signálů přicházení na vstup dříve, než přijde zpožděné slovo z posuvného registru.

Opravné signály totiž na výstup chodí v intervalech po patnácti krocích, a tudíž by mohly zapříčiňovat opravu nesprávných bitů vycházejících z dekodéru. Přivedením nuly na hradlo AND (v prvních patnácti krocích), které propouští opravný bit, jsem zamezil vyslání nechtěného opravného bitu.

### 10.3.3 VHDL návrh hradla EX-OR

```
library ieee ;
use ieee.std_logic_1164.all ;

entity EX_OR is
        port (x1, x2 : in std_logic ;
                    f        : out std_logic ) ;
end EX_OR ;

 architecture LogicFunction of EX_OR is
 begin
         f<= (x1 and not x2) or (not x1 and x2) ;
 end LogicFunction ;
```

## 10.4 Test funkčnosti Meggitova dekodéru

Pro simulaci funkčnosti jsem vytvořil ve „Vector Waveform“ soubor se čtyřmi vstupy PRN, U, CLK, CLR, které fungují podle tabulky č. 7 a jedním výstupem, který je znázorněn na předchozím obrázku úplně vpravo. Pro opravu jednoho patnáctibitového slova je potřeba 30 kroků dekodéru. Vstup „U“ představuje přijaté slovo w(x), které přichází do dekodéru.

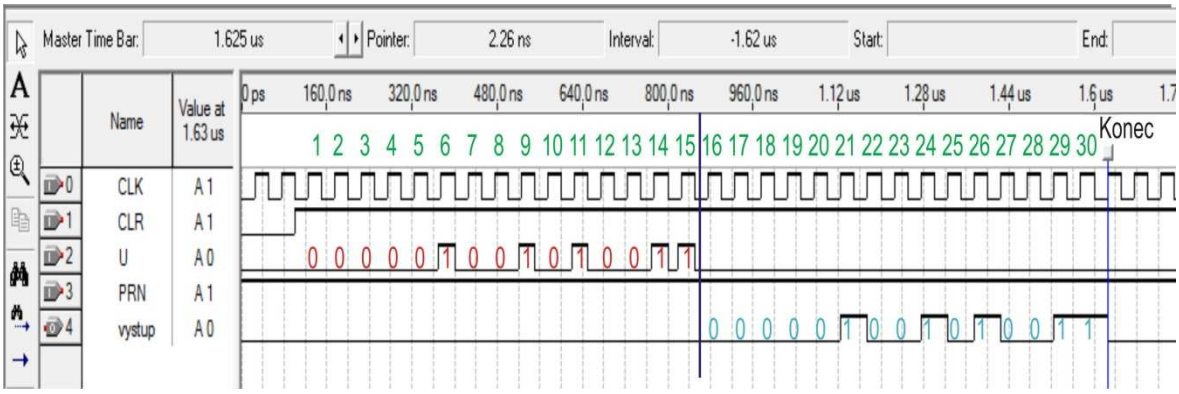
Testování Meggitova dekodéru jsem provedl na dvou příkladech. V prvním z nich jsem do obvodu poslal patnáctibitové přijaté slovo $w(x) = x^9 + x^8 + x^6 + x^4 + x + 1$ v binární podobě 000001101010011, které mělo chybu na pozici bitu $x^8$ a dekodér jej tedy musel opravit (červeně jsou znázorněny nulové nepoužívané bity).

Na následujícím obrázku je vidět, že Meggitův dekodér v prvních patnácti krocích slovo přijal a v následujících patnácti krocích toto slovo opravil na správné slovo ve tvaru $w(x) = x^9 + x^6 + x^4 + x + 1$ v binární podobě 000001001010011.


Obr. č. 23 – Oprava poškozeného slova Meggitovým dekodérem

Na předchozím obrázku jsou vidět zelené číslice, které označují kroky dekodéru. Dvě modré svislé čáry, které rozdělují simulaci na část výpočtovou a opravnou. Dále je červenými číslicemi znázorněné přijaté slovo a modrými číslicemi je znázorněno slovo opravené na výstupu v druhých patnácti krocích dekodéru.

Ve druhém testu funkčnosti dekodéru jsem na vstup tohoto obvodu poslal kódové slovo přijaté ve správném tvaru. Tudíž dekodér přijaté slovo poslal na výstup bez jakékoliv opravy. Slovo je ve tvaru $w(x) = x^9 + x^6 + x^4 + x + 1$ v binární podobě 000001001010011 a na výstupu tak také zůstalo.




Obr. č. 24 – Přijaté slovo bez opravy Meggitovým dekodérem

Z obrázku je patrné, že slovo přišlo bez chyby a tudíž se nemuselo opravovat. Simulaci jsem provedl na stejném návrhu obvodu jak tomu bylo v předchozím případě zobrazeném na obrázku 23.

Ze simulací vyplývá, že návrh a realizace Meggitova dekodéru dopadla podle očekávání dobře. Jednotlivé součástky zapojené do tohoto speciálního obvodu spolu vzájemně fungovaly a opravovaly příchozí slova. V posledním příkladu bylo vidět, že pokud přišlo slovo bez chyby, toto slovo se v pořádku dostalo na výstup dekodéru. Tyto simulace jsem prováděl jako funkční, a proto je zde zanedbáno zpoždění součástek a doba překlápění jednotlivých klopných obvodů.

# ZÁVĚR

Cílem této práce bylo pochopení a realizace digitální modulace za pomoci zákaznických hradlových polí FPGA. V první teoretické části jsem udělal několik rešerší z oblasti PCM digitálních modulací a jejich zrychlení pomocí adaptivních metod. U těchto zrychlení jde především o redukci bitů, které procházejí přenosovým prostředím. Diferenciální PCM kóduje rozdíly mezi okamžitou hodnotou vzorku signálu v daném vzorkovacím okamžiku a hodnotou predikovanou z předchozích vzorků. Její vylepšená verze ADPCM se dokáže přizpůsobit v počtu kontovacích hladin.

Dále jsem v práci rozebral převod lineárního na robustní kódování. Princip převodu závisí na redukci počtu bitů v přenášeném slově. Všeobecně platí, že čím méně bitů přenášíme, tím je menší pravděpodobnost chyby. Proto převod z 13-ti bitového vzorku na 8-mi bitový můžeme považovat za převod z lineárního na robustní kódování. Tento převod je dán A-křivkou zobrazenou na obr. č. 1.

Pro vypracování práce bylo nutné využití systematického cyklického (12,8)-kódu, který vychází ze systematického cyklického (15,11)-kódu. Tento zkrácený kód zanedbává největší informační bity. Tyto bity ovšem může zanedbat jen za předpokladu, že jsou nulové. Každý cyklický kód je jednoznačně určen svým generujícím polynomem g(x) a skládá se z násobků právě tohoto generujícího polynomu. Pro tuto práci byl zvolen generující polynom $g(x) = x^4 + x^3 + 1$.

V praktické části jsem provedl realizaci kodéru a dekodéru (12,8)-kódu simulované v programu Quartus II od firmy Altera. Simulace jsem prováděl jen funkčně a nikoliv časově s připojením konkrétního hradlového pole. Pro realizaci jsem se rozhodoval mezi vývojovým prostředím od firmy Altera a vývojovým prostředím od firmy Xilinx. Přednost dostalo prostředí Quartus II od firmy Altera, protože poskytuje veškeré mnou potřebné funkce zdarma. Firma Xilinx již od minulého roku neposkytuje v neplacené verzi WebPack vizualizační prostředí.

Realizace kodéru spočívala v tom, že výsledné slovo w(x), které vychází z kodéru je složeno z informačního slova u(x) a zbytku po dělení generujícím r(x) polynomem g(x). Tento zbytek je k informačnímu slovu přidán jako redundantní (zabezpečovací) část na výstupu kodéru. Pro vytvoření zbytku po dělení r(x) je využíván LFSR neboli „Linear Feedback Shift Register“. Tento obvod je složen ze čtyř klopných obvodů a dvou hradel XOR. LFSR jde realizovat dvěma metodami, a to Galoisovou a Fibonacciovou metodou.

Pro vytvoření LSFR jsem zvolil metodu Galoisovu se zapojením hradel XOR mezi jednotlivé klopné obvody. Tato metoda se mi zdála vhodnější. Pro realizaci celého kodéru bylo ještě potřeba do obvodu zapojit dva multiplexory a jeden adresový čítač, který mezi nimi přepíná. Celý obvod jsem navrhoval v prostředí Quartus II – Block diagram. Klopný obvod jsem použil předdefinovaný z knihoven programu, ale oba dva multiplexory a jeden adresový čítač jsem vytvořil v jazyce VHDL a poté převedl jako blokovou součástku do tohoto diagramu se zapojením. Následné propojení celého schématu z obrázku č. 15 jsem ověřil několika osmibitovými vstupními slovy. Obvod pro kódování funguje bez problémů pro kódy (12,8) jak je vidět na simulacích v bodě 9.6.

Realizaci dekodéru systematického cyklického kódu jsem provedl jako zapojení Meggitova dekodéru. Tento princip byl objeven Meggitem v roce 1960 a spočívá v tom, že syndrom přijatého slova w(x) a syndrom samostatného slova s chybou mají ekvivalentní zbytek po dělení, jsou-li děleny stejným generujícím polynomem g(x). Pokud nastane chyba v přijatém slově w(x), oprava tohoto slova se bude provádět v případě, že vypočítaný syndrom bude nenulový. Syndrom se zjistí tak, že provedeme dělení přijatého slova w(x) generujícím polynomem g(x). Pokud proběhne dělení bezezbytku, to znamená s nulovým syndromem, je přijaté slovo ve správném tvaru a není jej potřeba opravovat. Pro dělení polynomů jsem využil již dříve mnou vytvořený obvod LFSR (kapitola 9.2). Celý obvod jsem taktéž navrhoval v prostředí Quartus II – Block diagram. Klopné obvody byly použity stejné jako u kodéru. Součástky, které jsem musel vytvořit pro toto zapojení v jazyce VHDL byly: čtyřvstupé AND, dvouvstupé AND, speciálně upravený čítač pro čítání do 30 a hradlo EX-OR. Tyto součástky po napsání kódu byly převedeny jako blokové součástky pro návrh kodéru v blokovém schématu. Přesné zapojení kodéru je popsáno v bodu 10.2 i s uvedenými VHDL kódy jednotlivých součástek. Pro ověření funkčnosti dekodéru jsem zvolil dvě patnáctibitová slova, která jak vyplývá z bodu 10.4, byla dekodérem dekódována správně. Ke kodéru je ještě třeba říci, že pracuje ve třiceti krocích a nelze jej zkrátit. V prvních patnácti krocích načítá kódové slovo a v druhých patnácti krocích jej opravuje.

Závěrem bych chtěl říci, že vytváření této práce mi poskytlo mnoho cenných zkušeností z oboru systematických cyklických kódů a hradlových polí FPGA. Byl bych velice rád, kdyby někomu dalšímu posloužila jako zdroj cenných informací ohledně těchto témat.

# CONCLUSION

The aim of this work was the understanding and realization of digital modulation using custom FPGAs. In the first part of the theoretical chapter, I did several searches of the PCM digital modulation and their acceleration using adaptive methods. The acceleration is mainly based on the reduction of bits that pass through the transmission environment. Differential PCM encodes the differences between the actual value of the sample signal at the sampling time and the value predicted from previous samples. Its improved version called ADPCM is able to adapt in a number of quantization levels.

I also worked up how to transfer from linear to robust encoding. The principle of transfer depends on reducing the number of bits in the transmitted word. Generally, the fewer bits we transmit, the less likely errors occur. Thus, the transfer of 13-bit sample to 8-bit can be considered a transfer from linear to robust coding. This transfer is shown by A-curve in figure number one.

For the development work was necessary to use a systematic cyclic (12,8)-code, which is based on a systematic cyclic (15,11)-code. This abbreviated code neglects the largest information bits. These bits can be neglected, however, only provided that they are zero. Every cyclic code is uniquely determined by its generating polynomial g(x) and consists of multiples of such generating polynomial. For this work was chosen generating polynomial $g(x) = x^4 + x^3 + 1$.

In the practical part I have made the implementation of encoder and decoder (12,8)-code simulation in Quartus II program by Altera Company. I conducted the simulations only by their funcion, but I did not work with a specific gate array it. I had two options for development environments, one from Altera company and the other from Xilinx company. Preference was given to environment Quartus II from Altera Company, because it provides all the necessary features for free. Since last year Xilinx does not support the unpaid version of WebPack visualization environments.

Implementation of the encoder to the fact that the resulting word w(x), which is based on the encoder consists of an information word u(x) and the rest of the division of generating r(x) of polynomial g(x). The rest of the information is added as a redundant word (security word) of the output encoder. To create the rest of the division of r(x) I used LFSR - "Linear Feedback Shift Register". This circuit is composed of four flip-flop circuits and two XOR gates. LFSR can be created using two methods, Galois and Fibonacci

method. To create LSFR I chose a method using Galois XOR gates between the different flip-flops. This method seemed to me better understandable. For the implementing of the encoder was needed to connect the circuit with two multiplexers and one address counter, with switch between them. I designed the entire circuit in Quartus II environment - Block Diagram. For Flip-flop circuit, I used the built-in libraries of the program, but the two multiplexers and one address counter I created in VHDL and then transferred as a block diagram of the device. I have checked rightness of the connection in figure number using several eight-bit input words. Circuit for coding works without problems for codes (12.8) as shown in the simulations in section 9.6.

For the implementation of the decoder of a systematic cyclic code, I made a decoder using Meggit method. This principle was discovered by Meggit in 1960 using the fact that the syndrome of the received word w(x) and syndrome of the separate words with an error is equivalent if they are divided by the same generating polynomial g(x). If an error occurs in the received word w(x), correction of this word will be carried out if the calculated syndrome is nonzero. Syndrome is found by dividing received word w (x) by generating polynomial g(x). If the division is exact, it means zero syndrome, the word is received in proper form and it does not need repair. For division of polynomials I used my previously created LFSR circuit (chapter 9.2). I also projected the entire circuit in the Quartus II environment - Block diagram. Flip-flops were the same as used in the encoder. Parts that I had to create for the implementation in VHDL are: four-inputs AND, two-inputs AND, specially modified counter for counting to 30 and EX-OR gate. These devices have been transferred to a block component for the design of an encoder in the block diagram. Precise implementation of the encoder is described in section 10.2 along with VHDL codes of individual components. To verify the functionality of the decoder, I chose two fifteen-bit words which both comes from section 10.4 which were decoded correctly. The encoder is operating in thirty steps and this can´t be shortened. In the first fifteen steps it loads the code words and in the next fifteen steps it corrects them.

Finally, I would like to say that the creation of this work provided me with valuable experience in systematic cyclic codes and FPGAs. I would be very glad if this work served as a source of valuable information on these topics.

# SEZNAM POUŽITÉ LITERATURY

[1] Shannonův teorém z *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 17.6.2006, 18.3.2010 [cit. 2010-04-27]. Dost. z WWW: <http://cs.wikipedia.org/wiki/Shannon%C5%AFv_teor%C3%A9m>.

[2] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Ztrátová komprese řečového signálu, s. 258. ISBN 80-7300-134-9. Cit. str. 40.

[3] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Ztrátová komprese řečového signálu, s. 258. ISBN 80-7300-134-9. Cit str. 41.

[4] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Konstrukce cyklických kódů, s. 258. ISBN 80-7300-134-9. Cit. str. 110.

[5] HRDINA, Zdeněk, Ing. CSc.; VEJRAŽKA, František, doc. Ing. CSc. *Digitální radiová komunikace*. Vyd. 1. Ostrava: Vydavatelství ČVUT, 1994. Modulace a demodulace, s. 256. ISBN 80-01-01059-7. Cit. str. 42.

[6] HRDINA, Zdeněk, Ing. CSc.; VEJRAŽKA, František, doc. Ing. CSc. *Digitální radiová komunikace*. Vyd. 1. Ostrava: Vydavatelství ČVUT, 1994. Modulace a demodulace, s. 256. ISBN 80-01-01059-7. Cit. str. 44.

[7] PINKER, Jiří ; POUPA, Martin . *Číslicové systémy a jazyk VHDL*. 1.vydání. Praha : Ben, 2006. 352 s. ISBN 80-7300-198-5.

[8] HRDINA, Zdeněk, Ing. CSc.; VEJRAŽKA, František, doc. Ing. CSc. *Digitální radiová komunikace*. Vyd. 1. Ostrava: Vydavatelství ČVUT, 1994. Modulace a demodulace, s. 79. ISBN 80-01-01059-7.

[9] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Konstrukce cyklických kódů, s. 258. ISBN 80-7300-134-9. Cit. str. 103.

[10] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Konstrukce cyklických kódů, s. 258. ISBN 80-7300-134-9. Cit. str. 115.

[11] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Konstrukce cyklických kódů, s. 258. ISBN 80-7300-134-9. Cit. str. 215.

[12] VLČEK, Karel, prof. Ing. CSc., *Komprese a kódová zabezpečení v multimediálních kumunikacích*. Vyd. 2. Praha : Ben, 2004. Konstrukce cyklických kódů, s. 258. ISBN 80-7300-134-9. Cit. str. 118.

[13] KOLOUCH, Jaromír, Doc, Ing. CSc., Programovatelné logické obvody. druhé, doplněné a upravené. VUT Brno : MJ Servis s.r.o., Brno, 2005. 15 s. ISBN 80-214-2986-0.

[14] BODER, Brian. Altera and Xilinx Report : The Battle Continues. *Seeking alpha* [online]. 17. července 2008, č.1, [cit. 2010-05-10]. Dostupný z WWW: <http://seekingalpha.com/article/85478-altera-and-xilinx-report-the-battle-continues>.

[15] *Altera : About Us* [online]. 2010 [cit. 2010-05-11]. FPGA CPLD and ASIC from Altera. Dostupné z WWW: <http://www.altera.com/corporate/about_us/abt-index.html>.

[16] PECH, Jan, Ing . *Www.HW.cz* [online]. 1.2.2002 [cit. 2010-05-12]. Nebojte se FPGA. Dostupné z WWW: <http://hw.cz/Teorie-a-praxe/Dokumentace/ART365-Nebojte-se-FPGA.html>.

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

APCM Adaptive PCM.

ASIC Application Specific Integrated Circuit.

CCITT Comité Consultatif International Téléphonique et Télégraphique.

CLB Configurable Logic Block.

CPLD Complex Programmable Logic Device.

DPSK Differential Phase Shift Keying.

FPGA Field Progammable Gate Array.

IP Intelectual Property.

LFSR Linear Feedback Shift Register.

LUT Look-Up Table.

NRZ Non Return To Zero.

NRZI Non Return To Zero Inverted.

PCM Pulse Code Modulation.

PLD Programmable Logic Devices.

PSK Phase-Shift Keying.

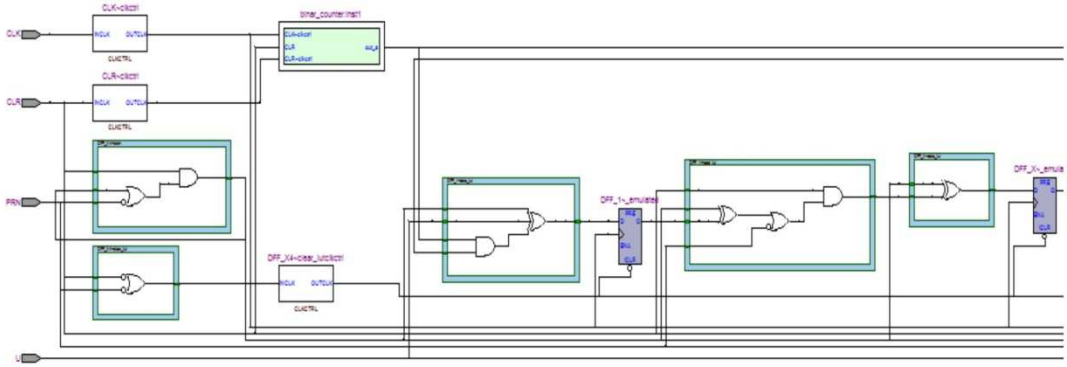
RZ Return To Zero.

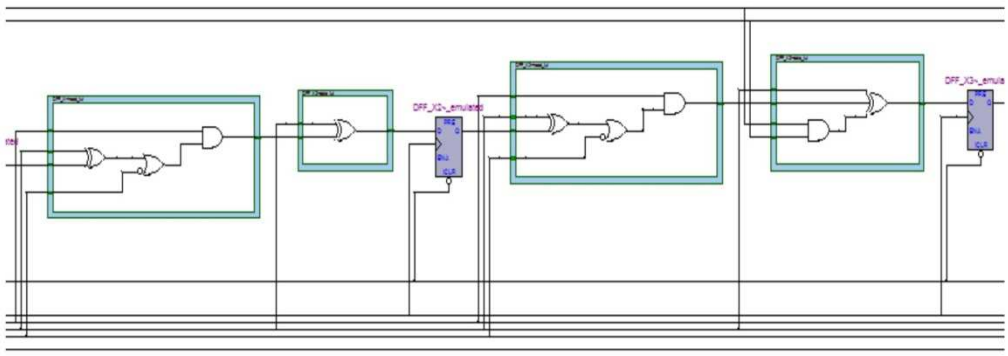
# SEZNAM OBRÁZKŮ

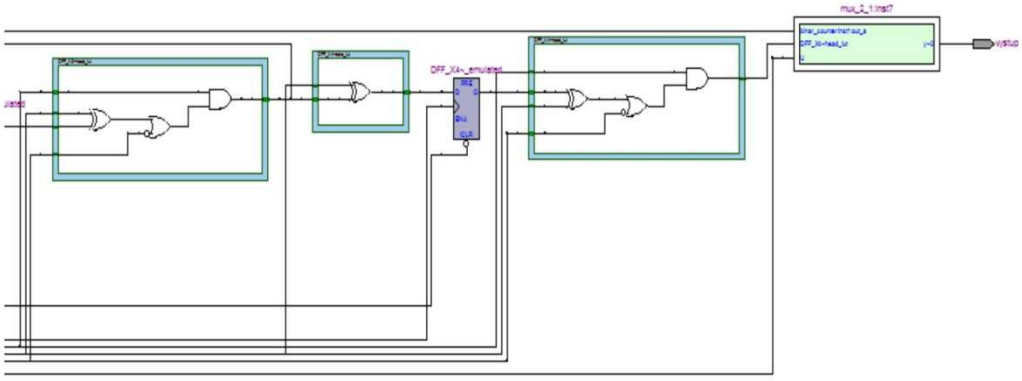
# SEZNAM TABULEK

# SEZNAM PŘÍLOH

# PŘÍLOHA P I: RTL SCHÉMA KODÉRU







# PŘÍLOHA P II: VHDL KÓD PRO ČÍTAČ DEKODÉRU

```
library ieee;
USE ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;

entity binar_counter_30 is
port (cl, res : in std_logic;
out_a : out std_logic );
end binar_counter_30;

architecture counter of binar_counter_30 is
signal a: bit_vector (0 to 4);
signal as: bit_vector (0 to 4);
        begin
                process (cl,res)
                        begin
                                if (res = '0') then
                                        a <= "00000";
                                        as <= "00000";
                                elsif (cl'event) and (cl = '1') then
                                        case a is
                                        when "00000" => as <= "00001"; a <= "00001";
                                                out_a <= '0';
                                        when "00001" => as <= "00010"; a <= "00010";
                                        when "00010" => as <= "00011"; a <= "00011";
                                        when "00011" => as <= "00100"; a <= "00100";
                                                                .
                                                                .
                                                                .
                                        when "01101" => as <= "01110"; a <= "01110";
                                        when "01110" => as <= "01111"; a <= "01111";
                                        when "01111" => as <= "10000"; a <= "10000";
                                                out_a <= '1';
                                        when "10000" => as <= "10001"; a <= "10001";
                                        when "10001" => as <= "10010"; a <= "10010";
                                        when "10010" => as <= "10011"; a <= "10011";
                                                                .
                                                                .
                                                                .
                                        when "11011" => as <= "11100"; a <= "11100";
                                        when "11100" => as <= "11101"; a <= "11101";
                                        when "11101" => as <= "11110"; a <= "11110";
                                        when others => as <= "00000"; a <= "00000";
                                                out_a <= '0';
                                        end case;
                                end if;
                        end process;
                end counter;
```

# PŘÍLOHA P III: OBSAH CDROMU

Na přiloženém CD se nachází text diplomové práce v různých formátech, dále projekty vytvořené v programu Quartus II 9.1 Web Edition pro kodér a zvlášť pro dekodér. Je zde také projekt pro děličku realizovaný taktéž v programu Quartus II. Jsou zde zařazena i testovací slova do jednotlivých projektů ve formě Vector Waveform File.

Přiložené CD obsahuje následující adresářovou strukturu:

.

| | |
|---|---|
| **./** Diplomová práce/ | text práce ve formátu PDF, DOC, DOCX |
| **./** Projekt kodéru/ | Projekt KODÉRU vytvořený v programu Quartus II |
| **./** Projekt dekodéru/ | Projekt DEKODÉRU vytvořený v programu Quartus II |
| **./** Projekt děličky/ | Projekt POLYNOMIÁLNÍ DĚLIČKY vytvořený v programu Quartus II |
| **./** Slova kodéru/ | soubory s příponou .vwf pro testování funkčnosti KODÉRU |
| **./** Slova dekodéru/ | soubory s příponou .vwf pro testování funkčnosti DEKODÉRU |
| **./** Slova dělička/ | soubory s příponou .vwf pro testování funkčnosti samostatného obvodu pro dělení polynomů |