AKCELERÁTOR PRO DEKÓDOVÁNÍ KONVOLUČNÍHO A REED-SOLOMONOVA ZABEZPEČOVACÍHO KÓDU

J. Kloub, A. Heřmánek

Ústav teorie informace a automatizace AV ČR, v.v.i.
Oddělení zpracování signálu.

Abstrakt

Při návrhu aplikací pro bezdrátový přenos dat je nutné simulovat přenosové kanály včetně vznikajících poruch při přenosu. Přenášená data jsou zatížena určitou chybou a kvalita jednotlivých opravných kódů je dána množstvím chyb, který je daný kód schopen opravit. Vzhledem k složitosti algoritmů pro kódování a především dekódování a opravu vzniklých chyb, může být simulace na běžných PC časově velmi náročná.

Časovou náročnost simulací lze snížit použitím hardwarových akcelerátorů. Akcelerátory využívají specifická obvodová řešení pro výpočet dílčích problémů simulovaného systému
a urychlí tak celkovou dobu simulace. V tomto článku je popsána implementace akcelerátorů pro dekódování a opravu dat zabezpečených konvolučním a nebo Reed-Solomonovým kódem. Akcelerátory jsou implementovány pomocí obvodů FPGA (Field-programmable gate array)
a pomocí signálového procesoru (DSP – pouze konvoluční dekoder).

1 Úvod

Rozvoj digitálních technologií přinesl možnost realizovat komplexní a výkonná řešení elektronických systémů. Výpočetního výkonu elektronických systémů je dnes využíváno v různých odvětvích při zpracovávání dat pomocí různých numerických metod.

Numerické metody umožňují například zabezpečit data proti chybám při přenosu komunikačním kanálem, který je zatížen šumem vedoucím k chybám v přenosu. Zabezpečení dat proti chybám opravnými kódy (FEC - Forward Error Correction) dnes hojně využívají například bezdrátové technologie jako WiFi, BlueTooth, DVB-T, WiMax a další. Mezi nejčastěji používané moderní kódu patří například blokový konvoluční a Reed-Solomonův kód.

Při návrhu aplikací pro přenos dat je nutné celý systém simulovat. Pokud je simulace spouštěna na standardním PC, jsou simulační kroky prováděny většinou sekvenčně v závislosti na schopnostech daného počítače. Vzhledem ke složitosti některých numerických metod může být simulace poměrně časově náročná.

Pro urychlení výpočtu simulace se nám nabízí realizovat některé výpočty pomocí akcelerátorů. V následujícím textu bude popsána realizace akcelerátorů pro dekódování konvolučního kódu a pro kódování a dekódování Reed-Solomonova kódu.

Struktura článku je následující: kapitola 2 popisuje prostředky pro realizaci akcelerátorů, kapitola 3 popisuje implementaci dekodéru konvolučního kódu, kapitola 4 popisuje implementaci Reed-Solomonova kodéru a dekodéru a kapitola 5 uvádí výsledky jednotlivých implementací akcelerátorů.

2 Prostředky pro HW implementaci akcelerátorů

Pro akcelerovaní výpočtů simulace lze použít různá obvodová řešení, jako například dedikované zákaznické obvody, mikroprocesory, signálové procesory, programovatelné obvody a jiná řešení.

Článek popisuje realizaci akcelerátorů pomocí obvodů s programovatelnými hradlovými poli (FPGA - Field-Programmable Gate Array) a s pomocí signálového procesoru (DSP - Digital Signal Processor).

Obvody FPGA dnes obecně disponují širokou škálou prostředků pro řešení různých typů výpočtů včetně zpracování signálů. Výsledné aplikace vzniknou pomocí propojení jednotlivých funkčních elementů obvodu. Jelikož se jedná o obvodové řešení, můžeme využít možnosti souběžného výpočtu datově nezávislých operací.

Signálové procesory bývají vystavěny na harvardské architektuře, kde je paměť programu oddělena od paměti dat. Tyto paměti mívají přístup realizován pomocí vlastních sběrnic, což zvyšuje propustnost systému. Další zvýšení výpočetního výkonu je zajištěno pomocí specializovaných jednotek, které dokáží pracovat paralelně. Jednou z jednotek bývá i rychlá násobička podporující vektorové operace (například operace typu $A \leftarrow A + B * k$, která provádí jeden krok skalárního násobení dvou vektorů v jednom taktu procesoru - přístup k vektorům je umožněn pomocí nezávislých adresních jednotek).

2.1 Realizace akcelerátoru pomocí obvodu FPGA

Pro realizaci akcelerátorů byly zvoleny vývojové desky (viz [3], [4], [5]) osazené FPGA firmy Altera: Stratix II, Stratix a Cyclone. Vývojové desky disponují ethernetovým rozhraním, které je využito k předávání dat mezi akcelerátorem a hostitelským systémem. Ethernetové rozhraní je na vývojových deskách implementováno pomocí obvodu SMSC LAN91C111 (viz [7]), který umožňuje komunikovat rychlostí 100 Mb/s (Full Duplex).

Data jsou mezi akcelerátorem a hostitelským systémem předávána pomocí protokolu UDP. Způsob připojení akcelerátoru je znázorněn na obrázku 1.

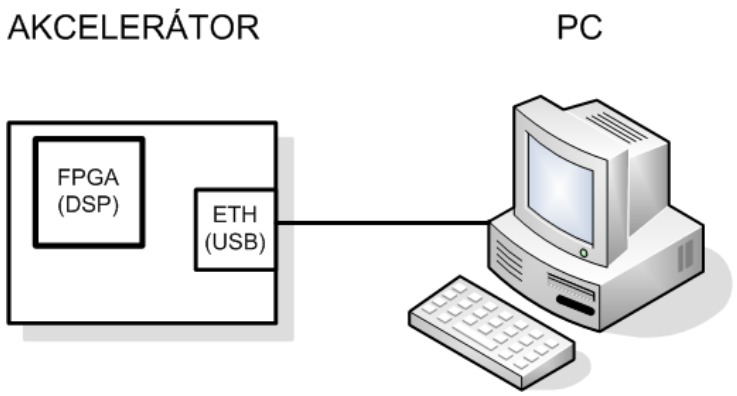
2.2 Realizace akcelerátoru pomocí DSP

V případě dekódování konvolučního kódu je pro akceleraci využit signálový procesor firmy Texas Instruments TMS320C6416, který disponuje koprocesorem pro Viterbiho algoritmus.

Pro implementaci akcelerátoru byla zvolena vývojová deska osazená zmíněným procesorem (viz [6]). Firma Texas Instrument vyvinula vlastní proprietární protokol RTDX (RTDX - Real Time Data Exchange) pro rychlou výměnu dat s vývojovou deskou přes rozhraní JTAG. Protokol RTDX je podpořen v prostředí Matlab (toolbox "Target for TI6000"). Na zvolené vývojové desce je rozhraní JTAG realizováno pomocí rozhraní USB. Data mezi DSP a koprocesorem jsou předávána pomocí DMA přenosů.

3 Implementace dekodéru konvolučního kódu

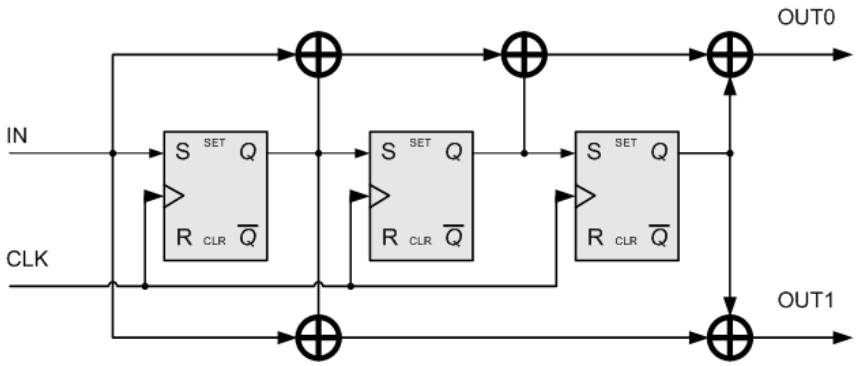
Konvoluční kódování dat lze velmi snadno realizovat obvodovým řešením. Základem kodéru je posuvný registr, který uchovává část z historie příchozích dat. Z této historie a z příchozích dat je odvozen výstup dekodéru podle generujících polynomů. Na obrázku 2 je znázorněno zapojení pro konkrétní parametry kódu, kde registry obsahují historii dat (stav kodéru). Výstupy jsou odvozeny pomocí součtu modulo-2 z hodnot určených polynomy. Potřebná datová redundance pro zabezpečení přenosu dat je odvozena z počtu výstupů kodéru, tedy počtem generujících polynomů.



Obrázek 1: Způsob propojení akcelerátoru a PC

Konvoluční kodér je charakterizován několika základními parametry:

- R = kódový poměr R = k/n , kde k je počet kódových bitů generující n výstupních bitů
- K = počet možných vazeb z posuvného registru do exkluzivních součtů
- G0(x) ... GN-1(x) = polynomy generující výstup



Obrázek 2: Příklad obvodové realizace konvolučního kodéru (K=4, R=1/2, G0=(1111), G1=(1101)

3.1 Dekódování konvolučního kódu a Viterbiho algoritmus

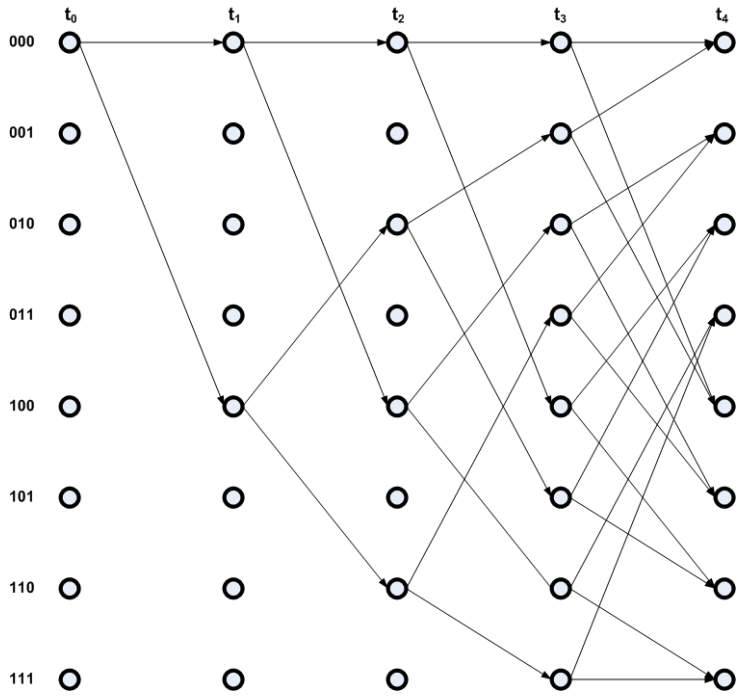
Pokud uvažujeme, že při přenosu došlo k chybě a nebo je použito takzvané děrování konvolučního kódu, musíme odhadovat nejpravděpodobnější zakódovanou bitovou posloupnost - hledáme tzv. "Most Likelihood" řešení.

Jedním z algoritmů pro určení nejpravděpodobnější bitové posloupnosti je Vitebiho algoritmus.

3.1.1 Viterbiho algoritmus

Viterbiho algoritmus se opírá o teorii grafů hledání nejlépe ohodnocené cesty. Graf je tvořen všemi možnými přechody mezi jednotlivými stavy historie konvolučního kodéru, tvořící mřížku (označováno také jako trellis). Vzhledem k tomu, že podstatou obvodové realizace konvolučního

kodéru je posuvný registr, lze velmi snadno určit strukturu přechodů v grafu. Na obrázku 3 je znázorněno jakým způsobem přechody v grafu vznikají (stavy jsou pro názornost označeny binárními hodnotami).



Obrázek 3: Expanze hran grafu mřížky v čase (pro K=4)

Graf začíná expandovat od počátečního stavu (stav 000). V každém dalším kroku expandují právě dvě hrany z dosažených stavů. Jedna hrana vede do stavu 1XX a druhá do stavu 0XX, kde XX jsou posunuté bity hodnoty stavu, z kterého hrany vychází (všimněme si, že nejvyšší bit následujícího stavu odpovídá zakódovanému bitu). Takto se tvoří kombinace cest, končící vždy v jednom ze stavů mřížky.

Pro výběr cesty, která odpovídá přijatým datům, musíme zavést pro jednotlivé hrany metriky. Ohodnoťme jednotlivé hrany hodnotou, která odpovídá hodnotě výstupu konvolučního kodéru ve stejném stavu jako je stav mřížky. Pro jeden stav kodéru existují dvě možné hodnoty výstupu v závislosti na vstupu kodéru. Pro hranu vedoucí do stavu s nejvyšším bitem rovným nule odpovídá ohodnocení pro vstup roven nule a obdobně i pro vstup rovný jedné.

V každém kroku vytváření grafu porovnáme ohodnocení hrany se vstupním symbolem. Metrika pro každou hranu je vypočítána jako kódová vzdálenost mezi ohodnocením hrany a vstupním symbolem v daném čase (např. Hammingova vzdálenost). Každá vytvořená cesta je ohodnocena součtem jednotlivých hranových metrik. Do jednoho stavu mřížky vedou maximálně dvě cesty. Cesta s horším ohodnocením může být "zapomenuta". Jsou-li ohodnocené cesty totožné, vybere se deterministicky jedna z nich (např. vždy ta cesta, která do stavu vede ze stavu s nižší hodnotou). V každém stavu je uchována hodnota ohodnocení vítězné cesty do stavu vedoucí (označována jako akumulovaná metrika).

Vlastní dekódování dat spočívá ve zpětném průchodu grafem po nejlépe ohodnocené cestě. Teoreticky se graf může tvořit v čase do nekonečna. V praxi je graf omezen na časové okénko dané délky. Při zpětném průchodu grafem určuje aktuální stav přímo výstupní hodnotu. Jak bylo popsáno výše, výstupu odpovídá nejvyšší bit dosaženého stavu. Pořadí bitů výstupu je dekódováno v opačném pořadí, než byla data dekódována a pořadí musí být prohozeno.

Viterbiho algoritmus je poměrně časově náročný, pokud je implementován pomocí softwarových prostředků. Hardwarová implementace dekodéru využívá souběžného vyhodnocování v mřížkovém grafu, a tak může být dekódování rapidně urychleno.

3.2 Hardwarová implementace Viterbiho dekodéru

Jak bylo uvedeno výše, byl Viterbi dekodér implementován pomocí obvodu FPGA a pomocí DSP firmy Texas Instruments.

Nejdříve uveďme základní blokové schéma dekodéru implementovaného v obvodu FPGA. Viterbi dekodér se dá rozdělit do několika základních bloků:

- Jednotka pro výpočet metrik jednotlivých hran grafu mřížky (BMU - Branch Metric Unit).
- Jednotka pro přičtení metrik hran k aktuálním cestám a výběr nejlépe ohodnocených cest končících v aktuálních uzlech grafu (ACSU - Add-Compare-Select Unit)
- Jednotka pro výběr uzlu (stavu) do kterého vede nejlépe ohodnocená cesta (BSU - Best State Unit)
- Jednotka pro zpětný průchod grafem pro určení odhadované dekódované posloupnosti (SMU - Survior Management Unit)

Na obrázku 4 je znázorněno blokové schéma dekodéru.

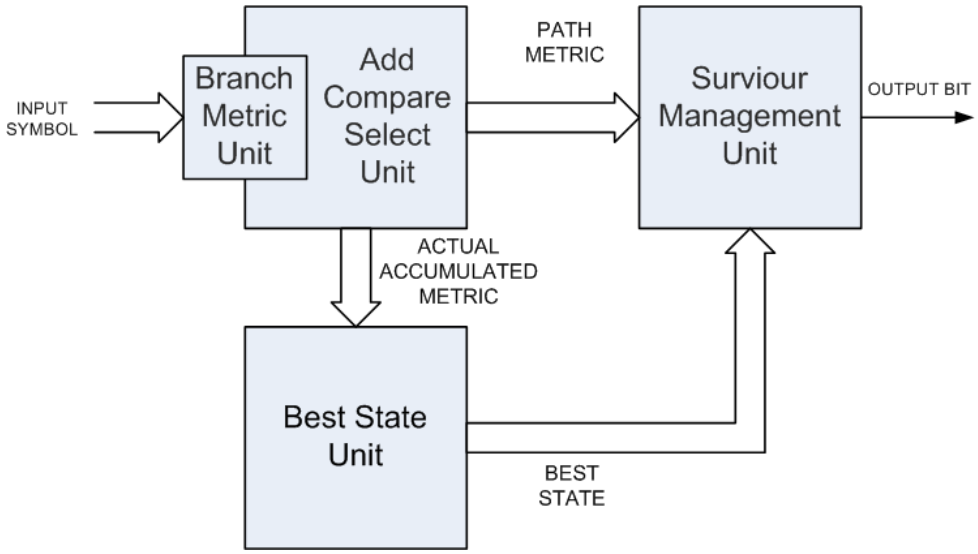
Jednotka BMU vypočítá metriky pro všechny nové hrany grafu na základě vstupního symbolu (Hammingova vzdálenost). Hranové metriky jsou předány jednotce ACSU, která přičte hodnoty k cestám vedoucích do aktuálních koncových uzlů grafu a vybere cesty s lepším ohodnocením pro každý koncový uzel grafu. Ohodnocení vybraných cest (akumulované metriky) si jednotka ACSU zapamatuje. Výstupem jednotky ACSU je informace o rozhodnutí výběru cest a akumulované metriky. Informace o rozhodování (označené jako "path metric") v průběhu dekódování jsou uchovávány v jednotce SMU.

Jednotka SMU má omezenou paměť (počtem kroků rozhodování - označme toto omezení "délkou" paměti). Délka paměti odpovídá délce zpětného průchodu ("traceback length"). Délku paměti lze nastavit pomocí parametru před syntézou zdrojových kódu implementace. Po naplnění paměti je jednotkou SMU proveden zpětný průchod od uzlu (stavu) grafu, který určí jednotka BSU na základě akumulovaných metrik (získané od ACSU). Výstupem jednotky SMU je dekódovaná bitová posloupnost.

V implementovaném dekodéru obsahuje jednotka SMU paměti dvě. Během zpětného průchodu jsou data čtena a vyhodnocovány z jedné paměti a do druhé paměti jsou současně zapisovány nové informace od jednotky ACSU. Po dokončení zpětného průchodu je účel pamětí zaměněn, a je tak docíleno průběžného dekódování příchozích dat.

Dekodér používá takzvané "hard" dekódování a jedná se o takzvané hybridní řešení. Hybridní řešení je kompromisem mezi plně paralelním vyhodnocováním všech uzlů v jednotce ACSU paralelně a sekvenčním vyhodnocením - vyhodnocovány jsou skupiny uzlů. Toto řešení vede ke snížení hardwarových nároků (zdroje obvodu, pracovní frekvence).

Popisovaná implementace Viterbiho dekodéru v obvodu FPGA vznikla v ÚTIA AV ČR (odd. Zpracování signálu).



Obrázek 4: Blokové schéma Viterbi dekodéru

3.2.1 Hardwarová implementace Viterbiho dekodéru pomocí DSP

Signálový procesor TMS320C6416 (DSP) je vybaven koprocesorem pro Viterbiho dekódování (VCP), který byl navržen pro bezdrátové standardy IS2000 a 3GPP jehož parametry jsou následující:

- Podpora dekódování pro K = 5, 6, 7 ,8 nebo 9.
- Uživatelsky zadávané koeficienty polynomů.
- Kódový poměry 1/2, 1/3 nebo 1/4.
- Možnosti nastavení délky a způsobu zpětného průchodu při dekódování.

Koprocesor provádí pouze vyhodnocování možných cest průchodu grafem a jeho zpětný průchod. Hranové metriky musí být vypočteny na straně DSP a jsou předávány pomocí dedikovaných DMA přenosů (EDMA - Enhanced Direct Memory Access). Pro správnou funkci dekódování je třeba nastavit jednotlivé přenosové kanály v řadiči EDMA. Řadič EDMA zahájí přenos na základě signálů od periferií a koprocesorů. Koprocesor generuje příslušné signály v případě prázdné vstupní paměti (bufferu) pro hranové metriky a v případě, že je výstupní paměť (buffer) vyplněna dekódovanými daty.

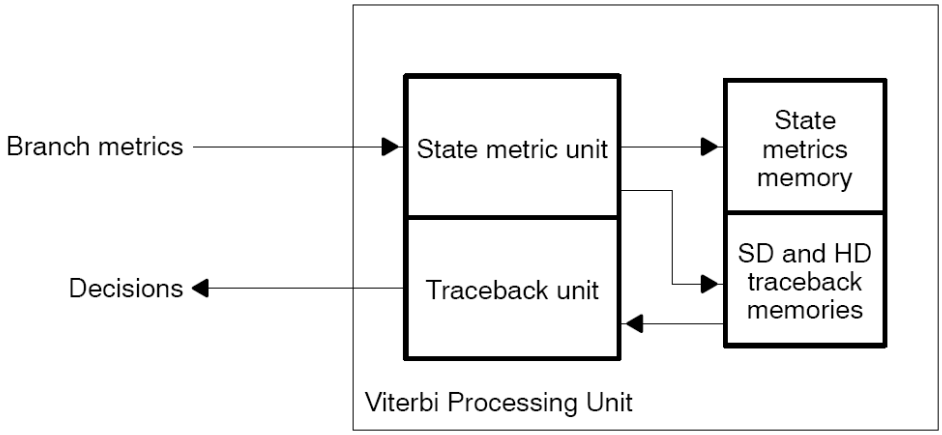
Jádro procesoru DSP pracuje nezávisle na koprocesoru a jeho pracovní frekvence je 1 GHz. Vlastní koprocesor pracuje na čtvrtinové frekvenci jádra, tedy na frekcvenci 250 MHz.

Na obrázku 5 je znázorněna architektura koprocesoru.

Bližší informace o koprocesoru lze nalézt v [2].

4 Implementace kodéru a dekodéru Reed-Solomonova kódu

Pro Reed-Solomonův kód byl implementován kodér i dekodér v obvodu FPGA. Reed-Solomonovy kódy používají operace nad Galoisovými tělesy (GF). Operace nad GF lze v obvodu FPGA implementovat efektivněji než pomocí standardních výpočetních jednotek. Převážně násobení GF



Obrázek 5: Blokové schéma koprocesoru pro Viterbiho algoritmus (převzato z [2])

je implementováno pouze pomocí hradel XOR a AND, namísto obecné násobičky. Operace nad GF na straně PC jsou velmi neefektivní.

4.1 Kodér Reed-Solomonova kódu

Kodér pro systematický RS kód může být realizován pomocí lineárního zpětnovazebního registru LFSR (obr. 6), který provádí dělení generujícím mnohočlenem podle vztahu (1).

$$x^{n-k}m(x) = q(x)g(x) + p(x) \tag{1}$$

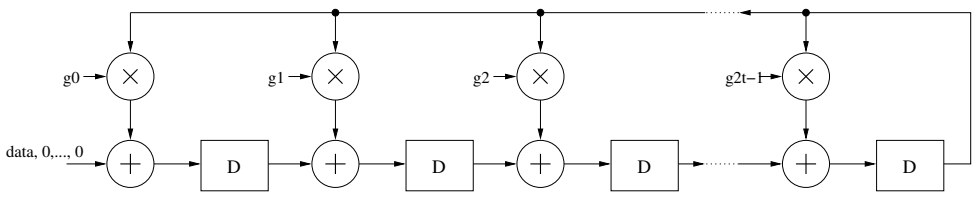
Na vstup registru přichází datové symboly určené k zakódování následované počtem $2t$ nulových symbolů (což odpovídá výrazu $x^{2t}m(x)$). Na vstup přichází koeficienty v pořadí $m_k, \ldots, m_1, m_0$. Všechny datové cesty na obrázku jsou osmibitové. Před zahájením dělení je potřeba vynulovat všechny registry. Po načtení všech n symbolů (tj. po n hodinových cyklech) budou registry obsahovat hodnoty odpovídající zbytku po dělení generujícím mnohočlenem.

K operaci zakódování jsou potřeba dvě operace nad konečným tělesem, a to sčítání a násobení. Sečtení dvou čísel v GF(256) je provedeno pomocí bitové operace xor. Násobení dvou čísel je provedeno následovně. Čísla z GF(256) lze vyjádřit pomocí mnohočlenu sedmého stupně s koeficienty z GF(2):

$$A(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0; a_i \in \mathrm{GF}(2), A \in \mathrm{GF}(256)$$

Operace $C = A \cdot B; A, B, C \in GF(256)$, lze vyjádřit v polynomiální reprezentaci jako

$$C(x) = A(x) \cdot B(x) = A(x) \times B(x) \mathrm{mod} f(x) = D(x) \mathrm{mod} f(x),$$



Obrázek 6: RS kodér používající dělění generujícím polynomem.

kde polynomi $A(x), B(x), C(x), D(x)$ jsou definovány jako:

$$\begin{aligned} A(x) &= a_7x^7 + a_6x^6 + \cdots + a_0, && (2)\\ B(x) &= b_7x^7 + b_6x^6 + \cdots + b_0, && (3)\\ C(x) &= c_7x^7 + c_6x^6 + \cdots + c_0, && (4)\\ D(x) &= d_{14}x^{14} + d_{13}x^{13} + \cdots + d_0 && (5) \end{aligned}$$

kde , $a_i, b_i, c_i, d_j \in GF(2)$, mod je operace výpočet zbytku po dělení polynomem a $\times$ je operace násobení polynomů. Uvedené vztahy pro násobení dvou čísel platí pro těleso GF(256) s primitivním polynomem $f(x) = x^8 + x^4 + x^3 + x^2 + 1$.

$$\begin{aligned} d_{14} &= a_7b_7\\ d_{13} &= a_6b_7 + a_7b_6\\ d_{12} &= a_6b_6 + a_7b_5 + a_5b_7\\ d_{11} &= a_6b_5 + a_5b_6 + a_7b_4 + a_4b_7\\ d_{10} &= a_7b_3 + a_6b_4 + a_3b_7 + a_4b_6 + a_5b_5\\ d_9 &= a_3b_6 + a_6b_3 + a_2b_7 + a_7b_2 + a_4b_5 + a_5b_4\\ d_8 &= a_3b_5 + a_7b_1 + a_1b_7 + a_2b_6 + a_5b_3 + a_6b_2 + a_4b_4\\ d_7 &= a_4b_3 + a_1b_6 + a_0b_7 + a_7b_0 + a_2b_5 + a_5b_2 + a_3b_4 + a_6b_1\\ d_6 &= a_6b_0 + a_4b_2 + a_2b_4 + a_1b_5 + a_0b_6 + a_3b_3 + a_5b_1\\ d_5 &= a_4b_1 + a_1b_4 + a_5b_0 + a_3b_2 + a_2b_3 + a_0b_5\\ d_4 &= a_1b_3 + a_2b_2 + a_0b_4 + a_4b_0 + a_3b_1\\ d_3 &= a_1b_2 + a_2b_1 + a_3b_0 + a_0b_3\\ d_2 &= a_0b_2 + a_1b_1 + a_2b_0\\ d_1 &= a_0b_1 + a_1b_0\\ d_0 &= a_0b_0 \end{aligned} \qquad (6)$$

V druhém kroku se provede určení zbytku po dělení primitivním mnohočlenem, pomocí kterého získáme mnohočlen sedmého stupně. Pomocí operace "nalezení zbytku po dělení" se provede zobrazení patnácti binárních koeficientů $d_{14}, \ldots, d_0$ na osm $(c_7, \ldots, c_0)$.

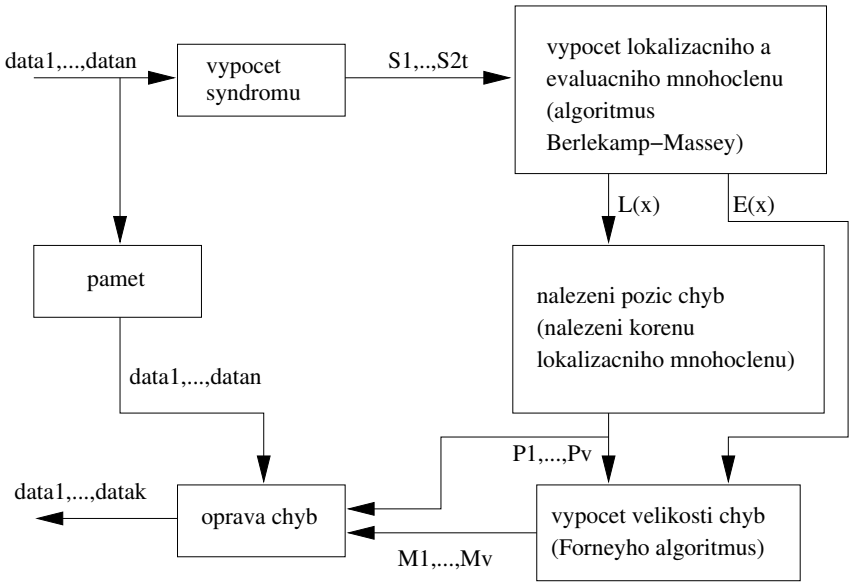
$$\begin{aligned} c_7 &= d_7 + d_{11} + d_{12} + d_{13}\\ c_6 &= d_6 + d_{10} + d_{11} + d_{12}\\ c_5 &= d_5 + d_9 + d_{10} + d_{11}\\ c_4 &= d_4 + d_8 + d_9 + d_{10} + d_{14}\\ c_3 &= d_3 + d_8 + d_9 + d_{11} + d_{12}\\ c_2 &= d_2 + d_8 + d_{10} + d_{12} + d_{13}\\ c_1 &= d_1 + d_9 + d_{13} + d_{14}\\ c_0 &= d_0 + d_8 + d_{12} + d_{13} + d_{14} \end{aligned} \qquad (7)$$

Násobení dvou čísel vyžaduje 77 hradel XOR, 64 hradel AND. V případě kodéru se provádí pouze násobení konstantou (tj. koeficienty generujícího mnohočlenu). Násobení konstantou vyžaduje méně hardwarových prostředků než obecná násobička. Například násobení číslem 59 vyžaduje pouze 28 hradel XOR.

4.2 Dekodér Reed-Solomonova kódu

Dekodér je založen na Berlekamp-Massey algoritmu pro určení lokalizačního a evaluačního mnohočlenu a na Forneyho algoritmu pro vyčíslení velikosti chyb. Tyto algoritmy tvoří kitickou

cestu algoritmu. Blokové schéma dekodéru je zobrazeno na Obrázku 7.



Obrázek 7: Blokové schéma RS dekodéru.

Příchozí symboly (na obrázku označeny jako $\texttt{data}_1, \ldots, \texttt{data}_n$ jsou uschovávány v paměti RAM. Symboly přicházejí v pořadí, v jakém vycházejí z kodéru, tj. nejprve datové symboly následované kontrolními symboly. Z příchozích symbolů se počítají hodnoty syndromů podle vztahu 8. Vzhledem k náročnosti implementace obecných mocnin v obvodu FPGA je nutné tyto vztahy převést podle Hornerova schématu. Tímto se úplně vyhneme mocnění, a tak lze výpočet velmi efektivně realizovat v FPGA. Stejná reprezentace vztahů je aplikován i v dalších krocích dekódování.

$$\begin{aligned} S_1 &= R(\alpha^0) = (e_0 + e_1 x + \cdots + e_{n-2}x^{n-2} + e_{n-1}x^{n-1})\,|_{x=\alpha^0} \\ S_2 &= R(\alpha^1) = (e_0 + e_1 x + \cdots + e_{n-2}x^{n-2} + e_{n-1}x^{n-1})\,|_{x=\alpha^1} \\ &\vdots \\ S_{2t} &= R(\alpha^{2t-1}) = (e_0 + e_1 x + \cdots + e_{n-2}x^{n-2} + e_{n-1}x^{n-1})\,|_{x=\alpha^{2t-1}} \end{aligned} \tag{8}$$

Z hodnot syndromů $S_1, \ldots, S_{16}$ se následně určí pomocí algoritmu Berlekamp-Massey (viz A) lokalizační a evaluační mnohočlen ($L(x)$ a $E(x)$). Po skončení algoritmu se pokračuje hledáním pozic chyb tak, že se postupným dosazováním všech prvků Galoisova tělesa do vztahu 9 hledají kořeny mnohočlenu. Hledání je tedy realizováno použitím takzvané "hrubé síly".

$$L(x) = 1 + l_1 x + l_2 x^2 + \cdots + l_{v-1}x^{v-1} + l_v x^v; x \in 1, 2, \cdots, 255 \tag{9}$$

Po nalezení pozic chyb se pokračuje výpočtem velikostí chyb pomocí Forneyho algoritmu. Oprava dat v paměti je provedena za podmínky, že 1) došlo alespoň k jedné chybě a 2) celkový počet chyb není větší než počet opravitelných chyb daného kódu.

Implementace kodéru a dekodéru byla vyvinuta v rámci diplomové práce [8].

5 Výsledky

Zde shrnujeme docílené výsledky implementace akcelerátorů pro podporu za prvé simulace dekódování blokového konvolučního kódu a za druhé simulace kódování a dekódování Reed-Solomonova kódu.

Výsledky implementace akcelerátoru pro dekódování konvolučního kódu

Výsledky implementací Viterbiho dekodéru jsou změřeny pro následující parametry dekódování:

- Použito "Hard" dekódování.
- Délka zpětného průchodu dekodéru je rovna délce dekódovaného rámce (bez použití sliding windows).
- K = 7
- R = 1/2
- G0 = 1111001
- G1 = 1011011

Graf na obrázku 8 ukazuje závislost doby dekódování na délce datového rámce pro jednotlivé implementace akcelerátorů.

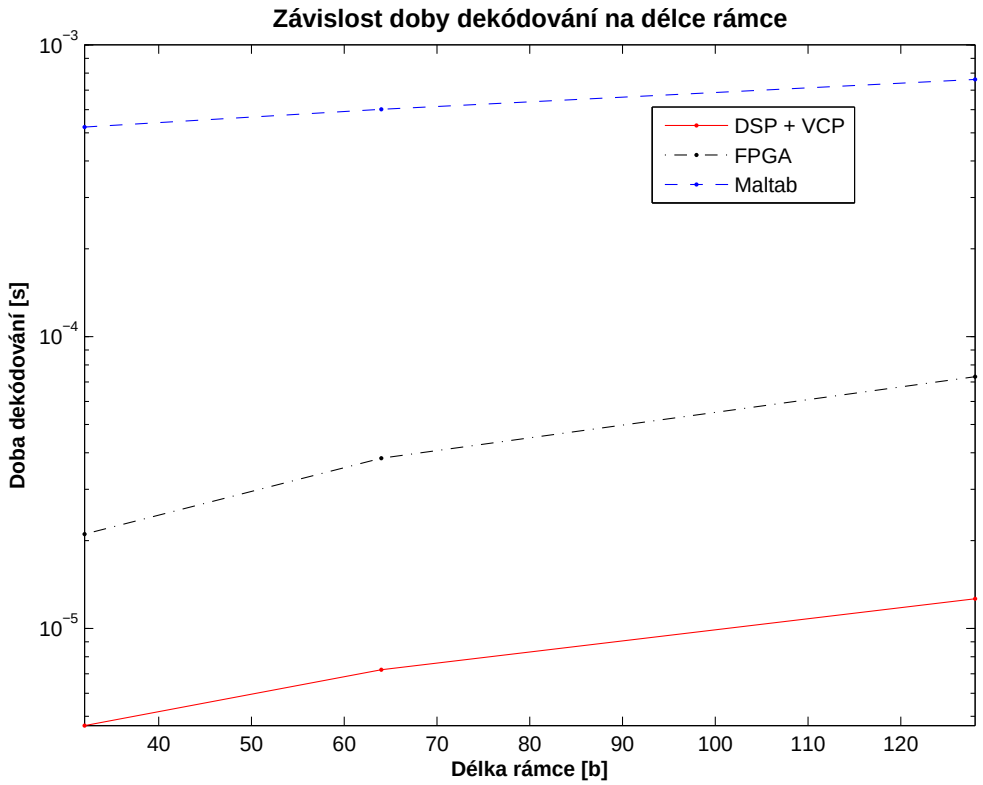
Hodnoty doby dekódování byly naměřeny v prostředí Matlab pomocí funkcí `tic` a `toc` (na PC, CPU Intel Core2 6600 @ 2.4 Ghz, 2 GB RAM; Windows XP SP2), doby dekódování v DSP pomocí čítače procesoru DSP (zahrnuto je výpočet hranových metrik na straně DSP, přenos pomocí EDMA a doba dekódování koprocesoru - viz [2]) a doby dekódování v obvodu FPGA jsou přímo určeny na základě znalosti struktury implementace dekodéru a její mezní pracovní frekvence.

Maximální pracovní frekvence implementace Viterbiho dekodéru na jednotlivých obvodech FPGA jsou uvedeny v tabulce 1. Hodnoty pracovních frekvencí jsou získány z reportů nástroje Quartus II. Obvod Cyclone nebyl pro implementaci Viterbiho dekodéru použit vzhledem k omezeným zdrojům obvodu.

Jak bylo zmíněno výše, používá implementace akcelerátoru ke komunikaci s hostitelským systémem rozhraní ethernet. Řízení ethernetového rozhraní je implementováno ve stejném obvodu FPGA jako vlastní dekodér. Maximální pracovní frekvence řízení ethernetového rozhraní je 50 MHz, a proto je snížena i pracovní frekvence dekodéru. Optimalizace akcelerátoru pro využití maximální možné pracovní frekvence dekodéru bude součástí další práce. Na grafu (obr. 8) je závislost doby dekódování v obvodu FPGA vztažena k pracovní frekvenci 50 MHz. Průměrná doba potřebná pro odeslání (příjem) jednoho ethernetového rámce byla změřena na zmíněném hostitelském sytému a její hodnota je 0,148 ms. V jednom datovém rámci je možné zaslat až 1272 bytů.

Obvod	**Max. pracovní frekvence**
Stratix II (EP2S180F1020C3)	147,74 MHz
Stratix (EP1S10F780C6)	100,65 MHz

Tabulka 1: Maximální pracovní frekvence implementace Viterbiho dekodéru v obvodu FPGA



Obrázek 8: Graf závislosti doby dekódování na délce rámce

Výkon a latence dekodéru lze zlepšit vzhledem ke známým omezením stávající implementace (optimalizace dekodéru je součástí budoucí práce).

Komunikace RTDX je zatížena poměrně velkou režií (doba obsluhy komunikačního kanálu) na zaslání jedné zprávy. Naměřená průměrná odezva dekódování jednoho rámce dat akcelerátorem přes RTDX je 0,94 s (součástí další práce bude optimalizace způsobu komunikace s akcelerátorem).

Výsledky implementace akcelerátoru pro kódování a dekódování Reed-Solomonova kódu

Doby kódování a dekódování byly naměřeny pro kód RS(255, 239, 8). V tabulce 2 a 3 jsou uvedeny doby kódování a dekódování v obvodu FPGA a v prostředí Matlab.

Stejně jako v případě akcelerátoru pro dekódování konvolučního kódu je pracovní frekvence akcelerátoru pro kodér a dekodér omezena na 50 MHz. Maximální pracovní frekvence kodéru a dekodéru jsou uvedeny v tabulkách

Jak je patrné z naměřených hodnot je hardwarová implementace Reed-Solomonova kodéru a dekodéru o několik řádů rychlejší než implementace v prostředí Matlab.

Poděkování: Tato práce vznikla za podpory projektů číslo 1ET100750408 a 1ET300750402 Grantové agentury AV ČR.

Doba kódování (FPGA)	**Doba kódování (Matlab)**
5,5 μs	1,2834 s

Tabulka 2: Doba kódování jednoho vstupního slova pro kód RS(255, 239, 8)

Doba dekódování (FPGA)	**Doba kódování (Matlab)**
37,4 μs	1,7178 s

Tabulka 3: Doba dekódování jednoho vstupního slova pro kód RS(255, 239, 8)

Obvod FPGA	f_{max}
Stratix II	138,51 MHz
Stratix	126,61 MHz
Cyclone	110,5 MHz

Tabulka 4: Maximální pracovní frekvence kodéru pro kód RS(255, 239, 8)

Obvod FPGA	f_{max}
Stratix II	98,35 MHz
Stratix	72,9 MHz
Cyclone	71,41 MHz

Tabulka 5: Maximální pracovní frekvence dekodéru pro kód RS(255, 239, 8)

Reference

[1] Hanzo, L.; Liew, T. H.; Yean, B. L. *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels*. John Wiley, 2002 ISBN 0-470-84726-3

[2] Texas Instruments. *TMS320C64x DSP, Viterbi-Decoder Coprocessor (VCP), Reference Guide* [online]. Dostupné na WWW:
`http://focus.ti.com/lit/ug/spru533d/spru533d.pdf`

[3] Altera. *Nios Development Board Reference Manual*, Cyclone Edition [online]. Dostupné na WWW:
`http://www.altera.com/literature/manual/mnl_nios2_board_cyclone_1c20.pdf`

[4] Altera. *Nios Development Board Reference Manual*, Stratix Edition [online]. Dostupné na WWW:
`http://www.altera.com/literature/manual/mnl_nios2_board_stratix_1s10.pdf`

[5] Altera. *Stratix II EP2S180 DSP Development Board Reference Manual* [online]. Dostupné na WWW:
`http://www.altera.com/literature/manual/mnl_stx2_pro_dsp_dev_kit_ep2s180.pdf`

[6] Texas Instruments. *TMS320C6416T DSK, Technical Reference* [online]. Dostupné na WWW: `http://c6000.spectrumdigital.com/dsk6416/V3/docs/dsk6416_TechRef.pdf`

[7] SMSC. *SMSC LAN91C111 Datasheet* [online]. Dostupné na WWW:
`http://www.smsc.com/main/datasheets/91c111.pdf`

[8] DUŠEK, Josef. *Návrh a implementace opravných kódů pro systém Orpheus*, Praha, 2007. Diplomová práce na ČVUT FEL. Vedoucí diplomové práce Ing. Martin Daněk, Ph.D.

Jan Kloub
ÚTIA AV ČR, v.v.i.
Pod Vodárenskou věží 4
Praha 8, 182 08
kloub@utia.cas.cz

Antonín Heřmánek
ÚTIA AV ČR, v.v.i.
Pod Vodárenskou věží 4
Praha 8, 182 08
hermanek@utia.cas.cz

A Algoritmus Berlekamp-Massey

Hodnota jak lokalizačního tak i evaluačního mnohočlenu může být nalezena iterativním způsobem, např. pomocí algoritmu Berlekamp-Massey. Původně, tak jak byl algoritmus navržen, sloužil jen pro výpočet lokalizačního mnohočlenu, ale lze ho použít i pro výpočet evaluačního mnohočlenu. Výpočet obou mnohočlenů pomocí tohoto algoritmu může být vyjádřen rekurentními rovnicemi:

$$\begin{aligned} d_i &= \textstyle\sum_{j=0}^{l} L_j^{(i-1)} S_{i-j} \\ l_i &= \delta(i - l_{i-1}) + (1-\delta) l_{i-1} \\ \begin{bmatrix} L^{(i)}(x) \\ A^{(i)}(x) \end{bmatrix} &= \begin{bmatrix} 1 & -d_i x \\ \delta d_i^{-1} & (1-\delta)x \end{bmatrix} \begin{bmatrix} L^{(i-1)}(x) \\ A^{(i-1)}(x) \end{bmatrix} \\ \begin{bmatrix} E^{(i)}(x) \\ C^{(i)}(x) \end{bmatrix} &= \begin{bmatrix} 1 & -d_i x \\ \delta d_i^{-1} & (1-\delta)x \end{bmatrix} \begin{bmatrix} E^{(i-1)}(x) \\ C^{(i-1)}(x) \end{bmatrix} \end{aligned} \tag{10}$$

Počáteční podmínky jsou

$$\begin{aligned} L^{(0)}(x) &= 1, \\ A^{(0)}(x) &= 1, \\ E^{(0)}(x) &= 0, \\ C^{(0)}(x) &= 1, \\ l_0 &= 0 \end{aligned} \tag{11}$$

Hodnota δ je definována jako

$$\delta = \begin{cases} 1, & \text{když } d_i \neq 0 \text{ a zároveň } 2l_{i-1} \leq i-1 \\ 0, & \text{jinak} \end{cases}$$

Symbol $L^{(i)}(x)$, resp. $E^{(i)}(x)$ je hodnota lokalizačního resp. evaluačního mnohočlenu v i-tém kroku, obdobně $A^{(i)}(x), C^{(i)}(x)$ jsou hodnoty pro pomocný lokalizační resp. evaluační mnohočlen, l_i je stupeň $L^{(i)}(x)$ v i-tém kroku, d_i je odchylka (v anglické literatuře označovaná *discrepancy*). Na základě odchylky d_i se určí nová hodnota $L^{(i)}(x), E^{(i)}(x)$. Lokalizační a evaluační mnohočlen jsou určeny po $2t$ iteracích, hodnoty se počítají pro $i = 1, 2, \ldots, 2t$.

Nalezení pozic chyb

Po dokončení výpočtu mnohočlenu $L(x)$ je potřeba nalézt jeho kořeny. To se provádí metodou hrubé síly v literatuře nazývané *Chien search*, která spočívá v tom, že se postupně dosazují za x všechny prvky z Galoisova tělesa GF(2^m) kromě nuly, tj. vyčíslují se hodnoty
$L(1), L(2), \ldots, L(2^m - 1)$ a zaznamenávají se nalezené kořeny mnohočlenu, tedy hodnoty, pro které platí $L(x) = 0$. Pro nalezení pozic chyb P_l je nutné kořeny lokalizačního mnohočlenu zinvertovat a zlogaritmovat, což znamená nalézt takové číslo k, pro které platí $P_l = \alpha^k$, tedy $\log(P_l) = log(\alpha^k) = k$.

Výpočet hodnot chyb

Po nalezení pozic chyb je ještě potřeba vypočítat hodnoty velikosti chyb z mnohočlenu $E(x)$. Výpočet se provádí pomocí *Forneyho algoritmu* (Forney Algorithm). Hodnota chyby $M_l, l \in$

$1, \ldots, v$ je určena obecně vztahem:

$$M_l = -\frac{E(P_l^{-1})}{P_l^{-1} L'(P_l^{-1})} \tag{12}$$

$L'(x)$ značí derivaci lokalizačního mnohočlenu. Derivace mnohočlenu $w(x)$ stupně v, $w(x) = w_0 + w_1 x + w_2 x + \cdots + w_v x^v$ je v GF(2^m) definována vztahem

$$L'(x) = w_1 + w_3 x^2 + w_5 x^4 + \cdots + v w_v x^{v-1} = \sum_{j=1}^{v} j w_j x^{j-1}$$

V případě RS kódu s kořeny generujícího mnohočlenu ve tvaru $\alpha^0, \alpha^1, \ldots, \alpha^{2t-q}$ lze rovnici 12 pro určení velikosti chyby M_l vyjádřit

$$M_l = \frac{P_l E(P_l^{-1})}{\prod\limits_{\substack{j=1 \\ j \neq l}}^{v} (1 - P_j P_l^{-1})} \tag{13}$$

Při výpočtu hodnoty M_i se do evaluačního mnohočlenu dosadí inverzní hodnota pozice l-té chyby a tato hodnota se vynásobí hodnotou pozice l-té chyby a následně podělí součinem výrazů přes všechny pozice chyb j rozdílné od pozice právě hledané velikosti chyby l.

Oprava chyb

Nyní jsou již známy všechny hodnoty potřebné k sestavení odhadu chybového mnohočlenu $\hat{\mathbf{e}}(x)$. Například pro hodnoty $(M_1, P_1) = (\alpha^a, \alpha^k)$ bude mít chybový mnohočlen $\hat{\mathbf{e}}(x)$ tvar $\hat{\mathbf{e}}(x) = \alpha^a x^k$. Chybový mnohočlen se sečte z mnohočlenem přijatého slova a získá se odhad $\hat{U}(x)$ původně vyslaného mnohočlenu $U(x)$

$$\hat{U}(x) = R(x) + \hat{e}(x) \tag{14}$$

Po odstranění kontrolních symbolů dostaneme odhad zdrojového mnohočlenu $\hat{m}(x)$

$$\hat{m}(x) = \hat{U}(x)/x^{n-k}$$