

Technická zpráva



Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

Demonstrátor konvolučního kodéru a Viterbi dekodéru s ethernetovým rozhraním implementovaný v FPGA

Ing. Jan Kloub, Ing. Antonín Heřmánek PhD.
{kloub, hermanek}@utia.cas.cz, +420-2-6605 2511

Obsah

1 Úvod	1
2 Konvoluční kodér	1
3 Dekódování konvolučního kódu	2
3.1 Viterbiho algoritmus	2
3.2 Hardwarová implementace Viterbi dekodéru	3
3.2.1 Příprava vstupních dat	3
3.2.2 "Soft" a "hard" dekódování	4
3.2.3 Bokové schéma dekodéru	4
4 Struktura zdrojových kódů Viterbi dekodéru a konvolučního kodéru	11
4.1 Konvoluční kodér	11
4.2 Viterbi dekodér	12
5 Výsledky	13
5.1 ToDo	14
5.1.1 Soft dekódování	15
5.1.2 Puncturing	16
5.1.3 Snížení latence výstupu	16
6 Ověření funkce	17
6.1 Ověření funkčnosti dekodéru	17
6.2 Ověření funkčnosti kodéru	18
7 Výpis obsahu CD-ROM	23

Revize

Revize	Datum	Autor	Popis změn v dokumentu
0	19.12.2007	Kloub	Vytvoření dokumentu
1			
2			
3			
4			

1 Úvod

Tento dokument popisuje HW implementaci konvolučního kodéru a Viterbi dekodéru, který dekóduje konvoluční kód a opravuje případné chyby vzniklé při přenosu zakódovaných dat. Konvoluční kodér a Viterbi dekodér je implementován v Jazyce Handel-C. V kapitole 2 je popsána obvodová realizace konvolučního kodéru a v kapitole 3. Struktura zdrojových kódů je popsána v kapitole 4. Dosažené výsledky implementace Viterbi dekodéru jsou posány v kapitole 5. Ověření funkce kodéru a dekodéru je popsána v kapitole 6.

2 Konvoluční kodér

Při přenosu dat může dojít chybám, které se v praxi často shlukují. Pro dostatečné zabezpečení přenosu dat lze použít zabezpečovací kódy. Určitou skupinu z nich jsou kódy konvoluční.

Konvoluční kódování dat lze velmi snadno realizovat obvodovým řešením.

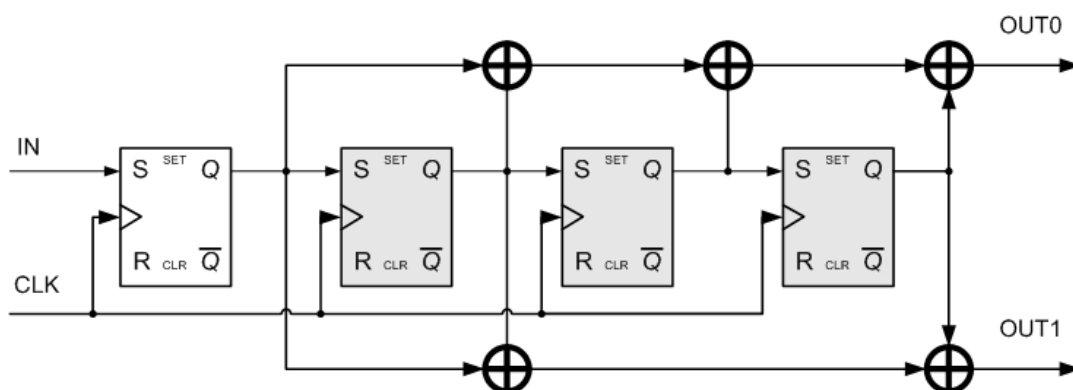
Konvoluční kodér je charakterizován několika základními parametry:

- N = počet výstupních bitů
- K = počet vstupních bitů
- L = počet paměťových registrů
- M = stupněm generujících polynomů + 1
- $Poly0(x) \dots PolyN-1(x)$ = polynomy generující výstup

V následujícím příkladu konvolučního kodéru bude popsána jeho obvodová realizace. Mějme následující parametry (popis způsobu volby a vhodnosti parametrů není součástí tohoto textu):

$$\begin{aligned} N &= 2 \\ K &= 1 \\ L &= 3 \\ M &= 4 \\ Poly0(x) &= x^4 + x^3 + x^2 + x \\ Poly1(x) &= x^4 + x^3 + x \end{aligned} \tag{1}$$

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 1 je znázorněno konkrétní zapojení pro zadané parametry, kde první registr zleva obsahuje kódovaná data a zvýrazněné registry 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: Příklad obvodové realizace konvolučního kodéru

Výstupy kodéru bývají serializovány do bitové posloupnosti. Pro zvýšení informačního poměru se v posloupnosti vynechávají některé bity podle stanoveného schématu (např. cyklicky pomocí pevně stanovené masky). Těto technice se říká děrování - puncturing (popis metody děrování není součástí tohoto textu).

3 Dekódování konvolučního kódu

Jak bylo popsáno v kapitole 2, může být datový přenos zabezpečen proti chybám konvolučním kódem. Pokud uvažujeme, že při přenosu došlo k chybě a nebo je použito takzvané děrování, musíme odhadovat nejpravděpodobnější zakódovanou bitovou posloupnost.

Jedním z algoritmů pro určení nejpravděpodobnější bitové posloupnosti je Vitebiho algoritmus. V dalším textu bude popsána hardwarová implementace dekodéru založená na tomto algoritmu.

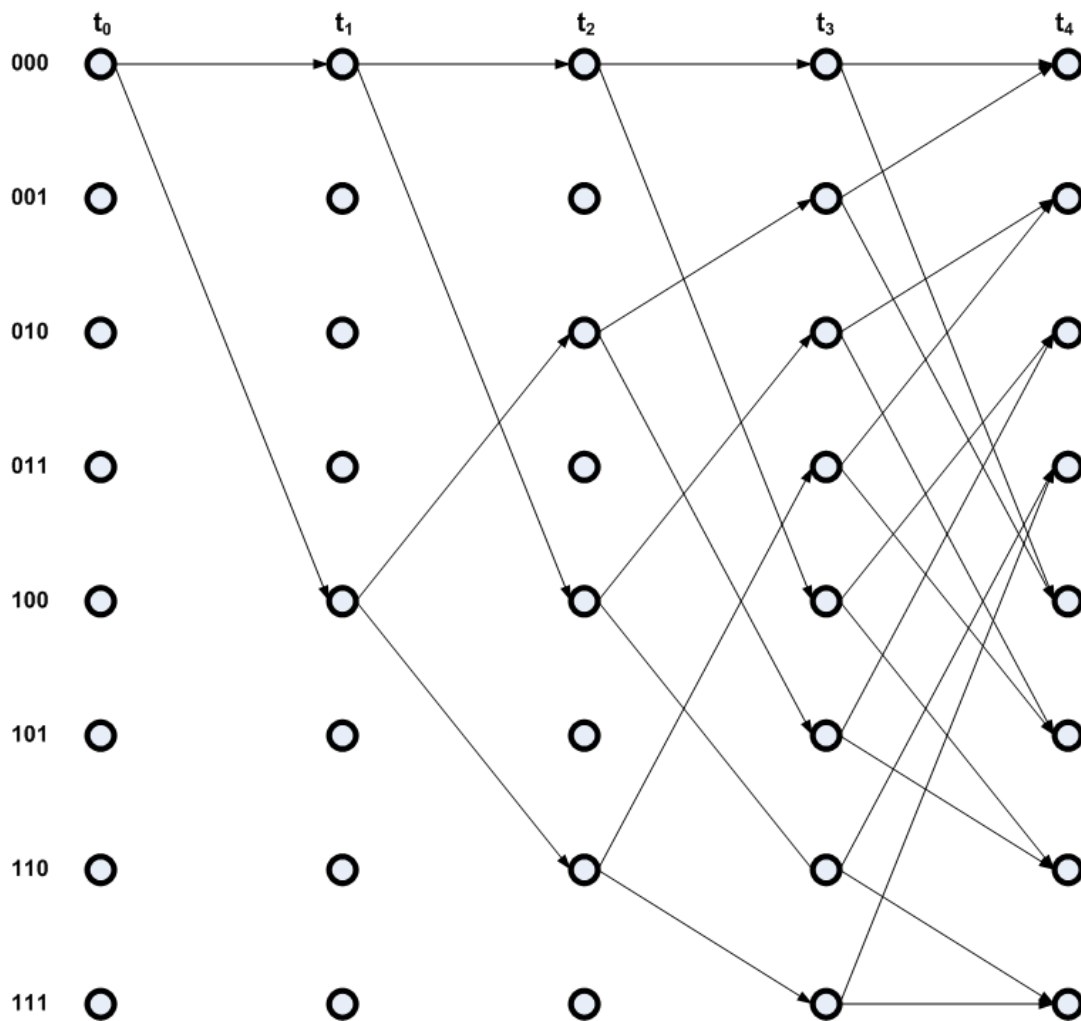
3.1 Vitebiho algoritmus

Vitebiho algoritmus se opírá o grafovou teorii 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 2 je znázorněno jakým způsobem přechody v grafu vznikají (stavy jsou pro názornost označeny binárními hodnotami).

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. Ohodnoňme 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. Hamingova 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).



Obrázek 2: Expanze hran grafu mřížky v čase (pro $L=3$)

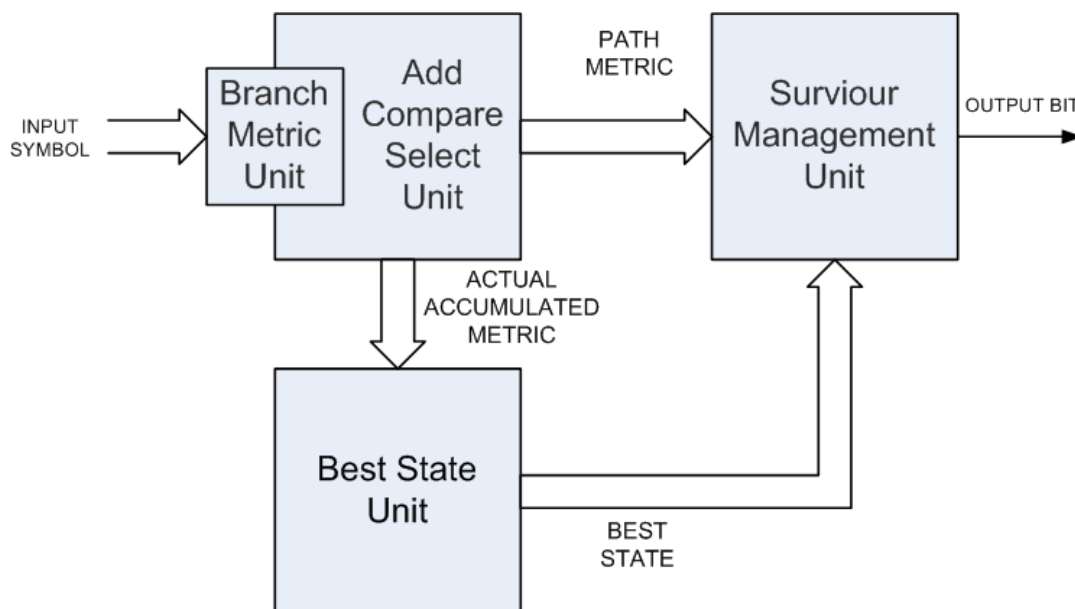
Vlastní dekódování dat spočívá ve zpětném průchodu grafem po nejlépe ohodnocené cestě. Teoreticky se graf může tvořit v čase do nekonečna. V praxi je graf omezen na časové okénko dané délky. Při zpětném průchodu grafem určuje aktuální stav přímo výstupní hodnotu. Jak bylo popsáno výše, výstupu odpovídá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

3.2 Hardwarová implementace Viterbi dekodéru

V následujícím textu popíši možnou hardwarovou implementaci Viterbi dekodéru a způsob reprezentace vstupních dat.

3.2.1 Příprava vstupních dat

Viterbi dekodér přijímá data zabezpečená konvolučním kódem. Jsou-li data serializována, musí být opět převedena do posloupnosti n -bitových slov původní reprezentace (viz. kapitola 2). Pokud bylo navíc použito děrování je nutné před převodem doplnit bitovou posloupnost o "prázdné" bity (většinou nuly). Tyto vložené bity nejsou pak zahrnuty do výpočtu metrik.



Obrázek 3: Blokové schéma Viterbi dekodéru

3.2.2 "Soft" a "hard" dekódování

V reálném světě jsou signály zatíženy šumem a jinými chybami. Jednotlivé vstupy dekodéru pro dekódování mohou nabývat různých úrovní. Pokud budeme uvažovat pouze dvě úrovně reprezentace jednotlivých vstupů dekodéru, tedy logická 0 a 1, hovoříme o tzv. "hard" dekódování. Při příjmu signálu bývají jednotlivé vstupy zachyceny A/D převodníky a jsou reprezentovány víceúrovňovou logikou. Víceúrovňovou logiku lze využít při výpočtu metrik grafu a je tak započítána i "síla" signálu.

Pro "hard" dekódování se pro výpočet metrik používá Hamingova vzdálenost a pro "soft" dekódování Euklidovská vzdálenost (popis výpočtu Euklidovské metrik není součástí tohoto textu a nadále bude popisován výpočet pomocí Hamingovi vzdálenosti).

3.2.3 Bokové schéma dekodéru

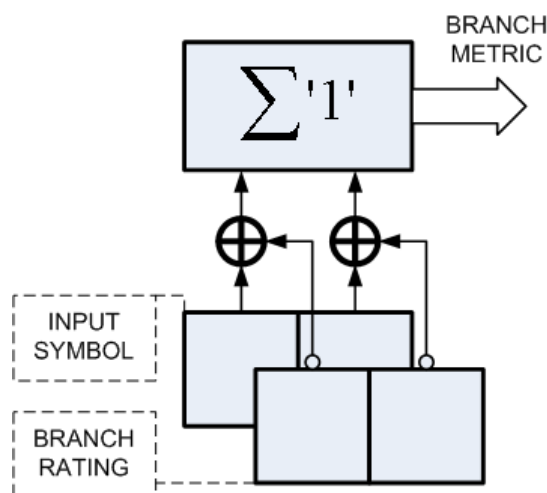
Viterbi se dá rozdělit do několika základních bloků:

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

Na obrázku 3 je znázorněno blokové schéma dekodéru.

BMU

Každá hrana grafu je ohodnocena odpovídající hodnotou výstupu konvolučního kodéru pro daný jeho stav (odpovídající stavu mřížky). Metrika hran je vypočítána na základě vstupního slova dekodéru. Zjednodušeně řečeno, hrana ohodnocená "nejpodobněji" k vstupnímu slovu má nejlepší metriku. Metrika lze



Obrázek 4: Možná hardwarová implementace výpočtu metriky

určit například pomocí Hammingovy vzdálenosti. Tento výpočet lze realizovat pomocí funkce XOR mezi jednotlivými bity ohodnocení hrany a vstupního slova dekodéru a součtem totožných bitů ve slovech. Vypočtené metriky jsou předány jednotce ACSU, které těmito hodnotami aktualizuje ohodnocení a výběr možných cest v mřížce grafu. Možná hardwarová implementace výpočtu metriky hran grafu je znázorněna na obrázku 4.

ACS

Jednotka označená jako ACSU přičítá hodnoty metrik nových hran v grafu k aktuálním cestám (prodloužení stávajících cest o jednu hranu). Ohodnocení prodloužované cesty je vždy uložena v předešlém stavu.

Každé dva stavy jsou propojeny s následujícími dvěma stavy takzvaným motýlkem, jak je vidět např. na obrázku 2. V posledním sloupečku grafu jsou vidět všechna propojení, tedy všechna úplná motýlková propojení.

Pro výpočet nových ohodnocení je nutné určit předešlé stavy propojené motýlkem, z kterých je přečteno předešlé ohodnocení cest. Předešlé stavy lze jednoduše určit následovně (stavy jsou označeny binární hodnotou): předchůdci stavu YXX jsou stavy XX0 a XX1, kde XX jsou posunuté bity označení aktuálního stavu doleva a Y je nejvyšší bit, který se "zahodí". Tato operace opět vyplývá z podstaty, že konvoluční kódér obsahuje posuvný registr.

Jak bylo již zmíněno, do následujícího stavu vedou maximálně dvě cesty. Počet cest do následujícího stavu je určen vzdáleností (počtem kroků v čase) od výchozího stavu (000) v mřížce. Cesty, které nezačínají ve výchozím stavu, neuvažujeme a nahradíme jejich ohodnocení nulou (nejnižším ohodnocením). Můžeme nyní uvažovat, že do následujícího stavu vedou vždy dvě cesty. Jednotka tedy vždy vybere jednu cestu do nového stavu. Výběry cest jsou předány jednotce SMU, kde jsou zapamatovány pro zpětný průchod grafem (pro každý krok a stav postačí 1-bitová informace výběru).

Protože historie grafu je ukládána v jednotce SMU, jsou v jednotce ACSU uchovány pouze akumulované metriky.

Chceme-li výpočet nových ohodnocení co nejvíce paralelizovat, pak pro každé motýlkové propojení existuje jedna výpočetní jednotka. V praxi to znamená, že čím více má mřížka stavů, tím více je třeba při implementaci hardwarových zdrojů. Z tohoto důvodu existují dvě rozdělení dekodérů - paralelní a hybridní. Paralelní obsahuje všechny výpočetní jednotky a hybridní je kompromis mezi sekvenčním a paralelním výpočtem, kdy je k dispozici jen určitý počet jednotek v jednom čase. Blokové schéma jedné výpočetní jednotky je znázorněno na obrázku 5. Hybridní struktura jednotky ACSU je znázorněna na obrázku 6.

Výpočetní jednotka má k dispozici ke každé hraně hodnotu metriky z jednotky BMU. Metrika je přičtena k ohodnocení celé cesty a uložena pro následný výběr nejlepší cesty v daném stavu. Je-li ohodnocení cest stejné musí být výběr deterministický (např. vybere cestu shora - horní paměťová buňka).

Jak je vidět z obrázků 5 a 6 obsahuje jednotka ACSU dva "sloupečky" paměťových buněk. V těchto buňkách jsou uloženy akumulované metriky. Nalevo znázorněné buňky obsahují předešlé hodnoty akumulovaných metrik a napravo znázorněné jsou buňky, do kterých jsou postupně ukládány nové hodnoty. Na konci výpočtu přes všechny motýlky, jsou nové hodnoty nahrazeny na pozice starých hodnot. Jelikož se metriky hran postupně přičítají, je nutné hodnotu akumulované metriky postupně "setřásat", aby nedošlo k přetečení.

Jednotka ACSU musí předat jednotce BSU nové hodnoty ohodnocení cest. Jednotka BSU určí, do kterého stavu vede nejlépe ohodnocená cesta a od něho je pak v jednotce SMU spuštěn zpětný průchod grafem.

Pokud je jednotka ACSU hybridní, je v každém kroku výpočtu pouze několik nových hodnot. Tyto hodnoty jsou předávány jednotce BSU, která postupně určí nejlepší stav. Na obrázku 6 je znázorněn krok výpočtu hybridní jednotky s dvěmi výpočetními jednotkami. Na obrázku určuje INDEX část mřížky, která se v daném kroku počítá. Paměťové buňky do kterých se ukládají nové hodnoty jsou na obrázku vyznačeny šedivě. Na obrázku je naznačeno, že jednotce BSU jsou aktuální hodnoty v každém kroku předány do vlastního pole paměťových buněk (vyznačeny šrafovaně), obsahující všechny nové hodnoty v jednom kroku výpočtu. Tím je eliminován složitý výběr aktuálních paměťových buněk v sloupci jednotky ACSU.

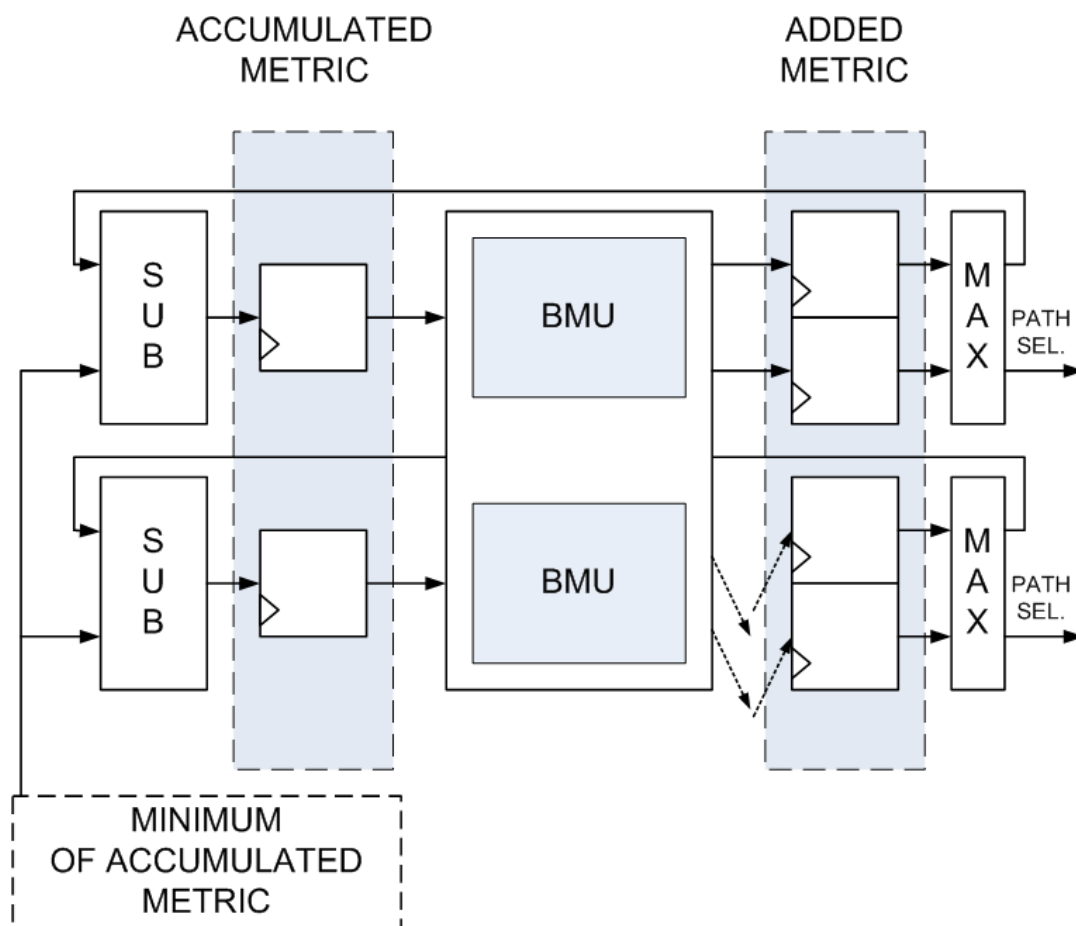
Shrneme-li výstupy jednotky ACSU, tak získáváme hodnoty pro určení nejlepšího stavu mřížky a hodnoty pro výběr cest. Hodnota nejlepšího stavu a historie výběru cest je pak použita při zpětném průchodu grafem.

BSU

Jednotka BSU určí stav mřížky, do kterého vede nejlépe ohodnocená cesta. Hodnota nejlepšího stavu se použije jako výchozí bod v mřížce pro zpětný průchod vytvořeného grafu. Vstupem do jednotky BSU je pole aktuálně vypočtených hodnot ohodnocení cest. Jednotka pracuje ve stejném počtu kroků jako jednotka ACSU. Jednotka určí index maximální hodnoty a její hodnotu ve vstupním poli (šrafované paměťové buňky). Aktuální nejlepší stav je určen z indexu ve vstupním poli a indexu určujícím krok výpočtu jednotky BSU. Hodnota aktuálního nejlepšího stavu a aktuální maximum je uloženo. V dalším kroku je předešlé maximum porovnáno s aktuálním. Bylo-li aktuální maximum větší než předešlé, pak je uloženo a je přepočtena hodnota nejlepšího stavu.

Úvaha při výpočtu stavu z indexu kroku jednotky a indexu vstupního pole:

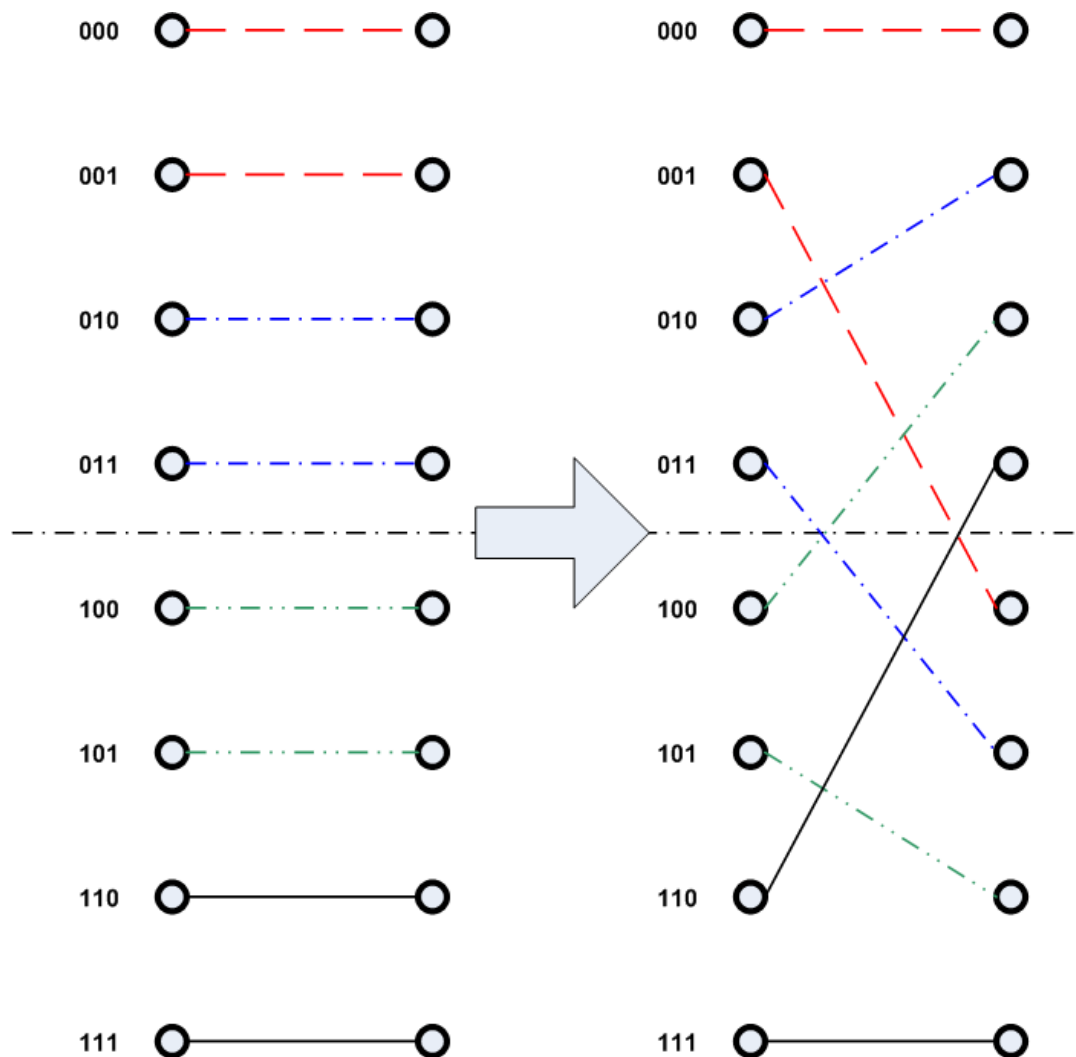
- Index kroku ukazuje bázi stavů - uvažujeme-li označení stavů ve sloupečku mřížky
- Určený index ve vstupním poli offset od báze - uvažujeme-li, že motýlek propojuje přímo stejné stavy
- Rozdělme stavy pro názornost na dvě poloviny - horní a dolní.
- V grafu je vždy jeden výstup motýlku v jedné polovině a druhý v druhé polovině - dáno vstupním bitem do posuvného registru kodéru (přijde-li 1 výstup je směřován do dolní poloviny stavů a naopak viz obrázek 7)
- Je-li vstupem motýlku lichý stav, pak výstupem je stav v dolní polovině mřížky a naopak
- Dochází tedy ke zhušťování výstupů do jedné příslušné poloviny mřížky - to odpovídá vysunutí nejméně významného bitu označení stavu



Obrázek 5: Struktura výpočetní jednotky ACSU

Z předešlých úvah lze přepočítat označení stavu pouhou rotací hodnoty součtu obou indexů. Přepočet určení stavu je znázorněn na obrázku 7. V některých literaturách je tato permutace adres označována jako "základní" (δ). Předpis základní permutace je následující (2):

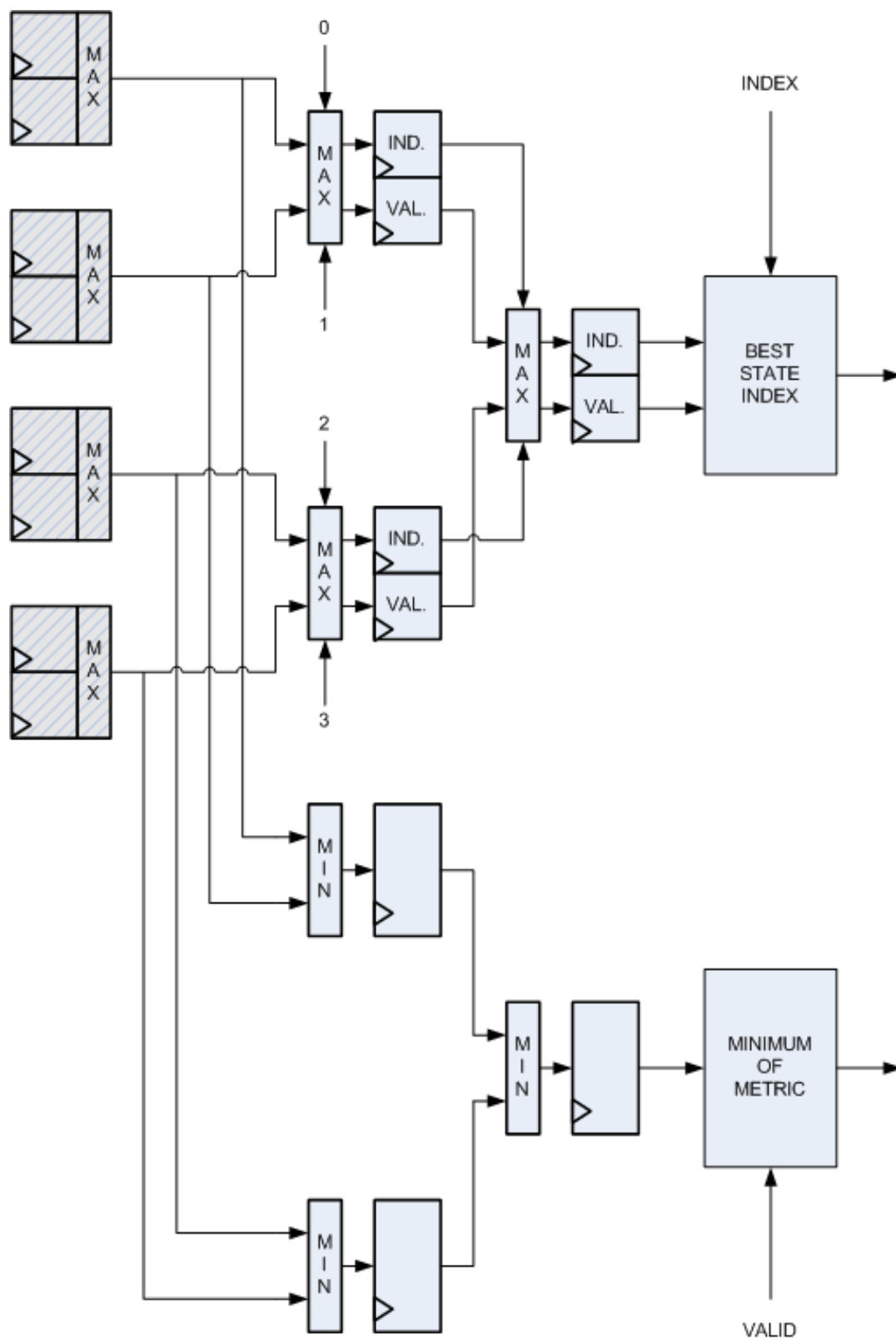
$$\delta_i(x_{n-1} \cdots x_{i+1} x_i x_{i-1} \cdots x_1 x_0) = x_{n-1} \cdots x_{i+1} x_0 x_i x_{i-1} \cdots x_1; i = L^2 \quad (2)$$



Obrázek 7: Transformace výstupů motýlků

Jednotka BSU zároveň hledá minimální hodnotu uloženou ve stavech jednotky ACSU. Minimum je použito pro setřásání hodnot v ACSU jako prevence proti přetečení uložených hodnot. Setřásání můžeme použít vzhledem k tomu, že nezáleží na absolutní hodnotě akumulované metriky, ale na relativní hodnotě - záleží jen na pořadí ohodnocení stavů. Jak je vidět na obrázku 8 je výpočet prováděn postupně v řetězci registrů (pipelining) vyváženého stromu. Pro výpočet minima je důležité, aby hodnoty v tomto řetězci byly již aktualizované předešlým setřásáním. Na obrázku 8 je naznačen signál VALID, který určuje, že je řetězec registrů již naplněn aktualizovanými hodnotami. Poté může začít hledání nového minima.

Shrneme-li výstupy jednotky BSU, tak získáme určení stavu (předáváno jednotce SMU) do kterého vede nejlépe ohodnocená cesta průchodu grafem a minimální hodnota akumulované metriky pro setřásání.



Obrázek 8: Struktura realizace jednotky BSU

SMU

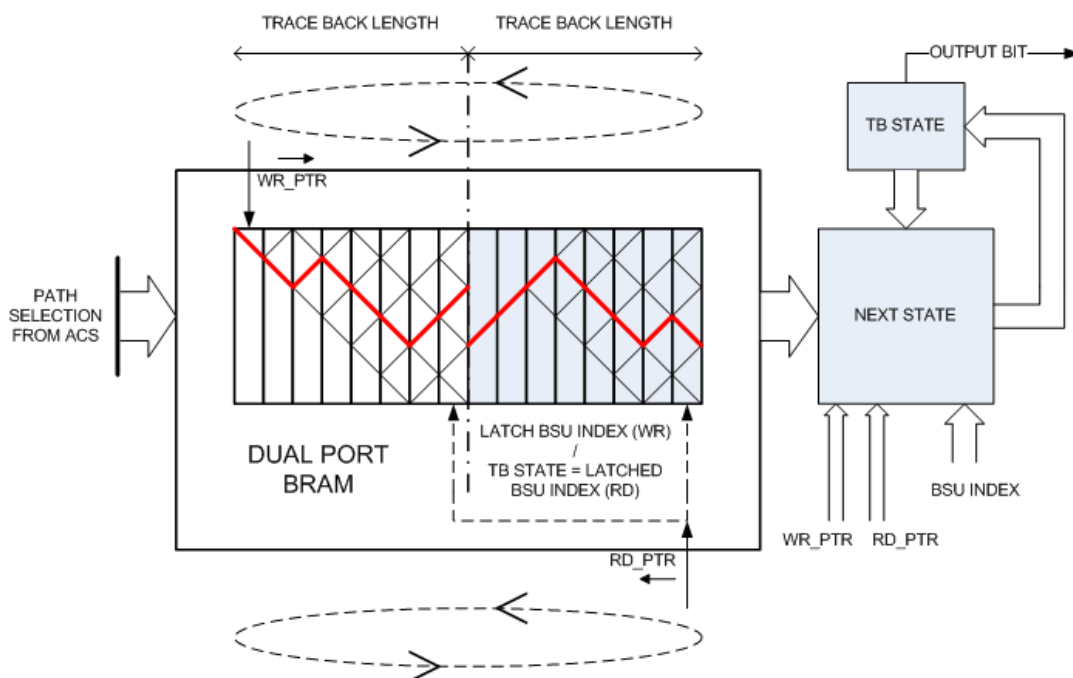
Jednotka SMU zajišťuje zpětný průchod grafem pro dekódování bitové posloupnosti. Jednotka získává historii rozhodnutí směřování v grafu od jednotky ACSU a stav, od kterého má být zahájen zpětný průchod od jednotky BSU.

Jednotka SMU je realizována pomocí dvoubranové paměti do které je zapisována a zároveň čtena informace o rozhodování průchodu grafem.

Délka ukládání historie rozhodování je omezena na časové okénko několika kroků průchodu grafem. V dvoubranové paměti jednotky je prostor pro dvě okénka. V jednom kroku je do jednoho okénka zapisována historie a z druhého je čtena. Po přečtení/zápisu celého okénka je účel okének prohozen (čtení je nahrazeno zápisem a naopak). Tohoto mechanismu lze snadno implementovat dvěma ukazateli do paměti. Ukazatel pro zápis historie je cyklicky inkrementován přes obě okénka od nulové adresy paměti. Ukazatel pro čtení historie je cyklicky dekrementován přes obě okénka od maximální adresy okének v paměti. Lze si takto představit, že okénka jsou spolu spojeny do válce a ukazatele rotují kolem něho v navzájem opačném směru.

Na obrázku 9 je znázorněna struktura jednotky s dvoubranovou pamětí.

Je zřejmé, že při startu dekódování jsou v okénku pro čtení neúčinná data a výstup je proto neplatný. Výstup je platný až po prvním "přepnutí" okének.



Obrázek 9: Struktura jednotky pro zpětný průchod grafu

Důležitými okamžiky jsou body, kdy se ukazatele kříží (hranice okének). Před tímto okamžikem je na straně zapisovací logiky zapamatován nejlepší stav z jednotky BSU. Po překřížení je tento stav výchozím stavem zpětného průchodu čtecí logiky. Zpětný průchod je pak odvozen od tohoto stavu a další stav průchodu v grafu je již určen obsahem paměti. V paměti je uložena informace o výběru cesty z jednotky ACSU. Jak bylo již zmíněno, v jednom stavu končí vždy dvě cesty a víme, které dva stavy jsou předchůdcem stavu stávajícího. Při zpětném průchodu postačí 1-bitová informace, která určí o jaký stav se jedná. Pro každý stav a každý krok průchodu je v paměti uloženo slovo s těmito bitovými informacemi. Pozice bitu v přečteném slově historie je určena přímo hodnotou aktuálního stavu průchodu. Na obrázku 10 je znázorněno jakým způsobem je získán další stav průchodu.

Z aktuálního stavu průchodu můžeme určit přímo jeden bit výstupu, což vyplývá z vlastnosti po-

suvného registru. Výstupem je nejvyšší bit hodnoty určující aktuální stav. Při zpětném průchodu je pořadí bitů otočené a musí být převedeno pomocí paměti typu LIFO pro každé okénko. Na obrázku 11 je znázorněn princip zápisu a čtení do paměti LIFO.

Pro adresaci obou pamětí je použit pouze jeden ukazatel, který je střídavě inkrementován a dekrementován (lze přirovnat k pohybu pístu motoru). Zápis a čtení je mezi jednotlivými paměťmi zaměňován po naplnění či vyčtení hodnot.

4 Struktura zdrojových kódů Viterbi dekodéru a konvolučního kodéru

Zdrojové kódy konvolučního kodéru a Viterbi dekodéru jsou napsány v jazyce Handel-C. Kodér i dekodér jsou napsány jako makro procedury. Parametry a některé makroprocedury pro kodér a dekodér jsou generovány pomocí skriptu v prostředí MATLAB (`congenerate.m`). Skript vygeneruje potřebné konstanty a makro procedury použité v kódu v závislosti na parametrech:

N Počet výstupních bitů konvolučního kodéru.

M Počet registrů konvolučního registru, včetně vstupního registru (viz obr. 1).

Poly0...N-1 Pole celočíselných hodnot určující jednotlivé polynomy genrující výstup kodéru.

ACS Počet jednotek ACS Viterbi dekodéru (tato hodnota musí být rovna mocnině 2 a nanejvýše 2^{M-2})

TBLENGTH Počet kroků zpětného průchodu Viterbi dekodéru (tato hodnota musí být sudá).

Výsledkem spuštění skriptu v MATLABu je hlavičkový soubor jazyka Handel-C (`convtab.hch`), který je do zdrojových kódů kodéru a dekodéru načten pomocí direktivy `#include`.

Skript je uložen na přiloženém CD (`\matlab\congenerate.m`)

4.1 Konvoluční kodér

Implementovaná makro procedura konvolučního dekodéru se jmenuje `conv_encoder()` a má tři vstupní parametry a jeden výstupní parametr.

d_in Jednobitový vstupní parametr.

din_ena Vstup určující platnost dat na vstupu.

d_out N-bitový výstup kodéru.

cnv_reset Reset kodéru.

Pokud je vstup `din_ena=1`, pak je každý hodinový takt načtena hodnota ze vstupu kodéru a je generován příslušný výstup. Výstup `d_out` je N-bitová hodnota, kde každý bit této hodnoty přísluší výstupu generovaným jedním polynomem konvolučního kodéru.

Stav kodéru lze resetovat pomocí parametru `cnv_reset`.

Hlavička makro procedury konvolučního kodéru je následující:

```
macro proc conv_encoder(d_in, din_ena, d_out, cnv_reset);
```

4.2 Viterbi dekodér

Makro procedura popisující chování Viterbi dekodéru se jmenuje `viterbi3hybrid()`. Na obrázku 12 je znázorněna makroprocedura s jejími vstupy a výstupy.



Obrázek 12: Znázornění vstupů a výstupu Viterbi dekodéru

Parametry a význam makro procedury jsou následující:

datain Datový vstup pro N-bitový symbol dekodéru.

datain_rdy Výstup určující připravenost dekodéru k příjmu jednoho vstupního symbolu. Tento výstup je držen v logické úrovni rovné 1 po dobu než vstup *datain_ena* potvrdí platnost vstupního symbolu. Během zpracovávání vstupního symbolu je tento výstup nastaven na hodnotu logické úrovně rovné 0.

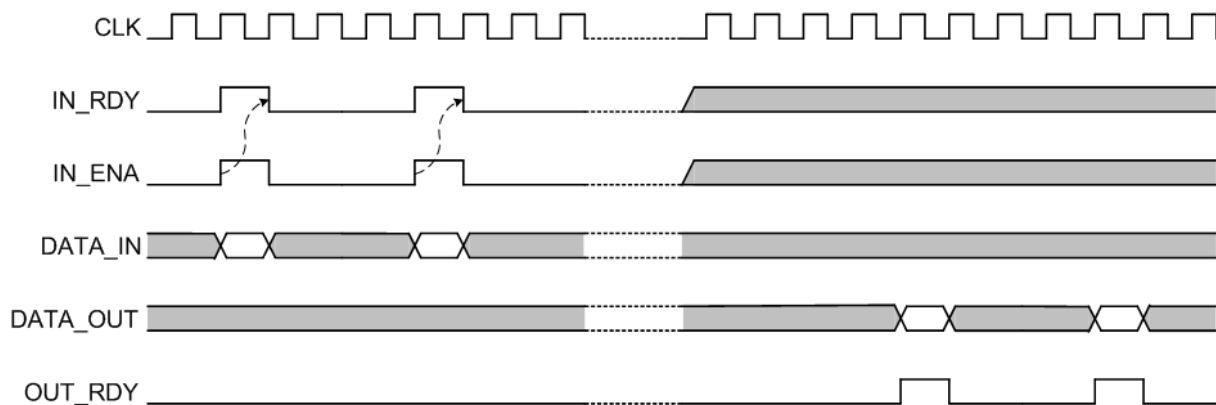
datain_ena Vstup určující platnost dat na vstupu.

dataout Jednabitový výstup dekodéru.

dataout_rdy Výstup určující platnost výstupních dat dekodéru.

rst Reset dekodéru. Dekodér je po resetu opět připraven po dvou hodinových cyklech, potřebných k nastavení výchozího stavu všech jednotek dekodéru.

Na obrázku 13 je znázorněno časování dekodéru. Výstupní data jsou platná až po přijetí takového počtu symbolů, kolik je dvojnásobek délky zpětného průchodu (kroky zpětného průchodu + kroky obrácení pořadí bitů). Každý další výstupní bit je pak platný s každým příchodem vstupního symbolu. Chceme-li vyčíst všechna dekodovaná data, musíme na vstup dekodéru přivést o odpovídající počet symbolů více, než bylo v zakódované posloupnosti.



Obrázek 13: Časování Viterbi dekodéru

Hlavička makro procedury Viterbi dekodéru je následující:

```
macro proc viterbi3hybrid(datain, datain_rdy, datain_ena, dataout, dataout_rdy, rst);
```

5 Výsledky

Údaje o časových omezeních a využití prostředků obvodů FPGA (Stratix 1S10, StratixII 2S180 a Virtex-4) byly určeny pomocí nástrojů Synplify, Quartus a ISE. Pro zmíněných určení vlastností byl použit referenční návrh, který obsahuje pouze dekodér a jeho vstupy a výstupy jsou přímo připojeny na vstupy a výstupy obvodů FPGA.

Dekodér je implementován v jazyce Handel-C. Syntéza kódů v Handel-C pomocí nástroje DK se ukázala jako neefektivní a byla použita syntéza pomocí nástroje Synplify. Při použití nástroje Synplify byly v závislosti na použitém obvodu FPGA ušetřeno až 60% zdrojů obvodu FPGA a ke zvýšení pracovní frekvence o 30%.

Nástroj:	DK(VHDL) + Synplify	Synplify + Quartus	DK(EDIF)	DK (VHDL) + Quartus
Log. zdroje	4266 LUTs	4173 LEs	10671 LUTs	3336 LEs
Paměť	x	5376 bit	5376 bit	5376 bit
Počet registrů	1956	x	1992	x
Časování	128,8 MHz	100,65 MHz	38,9 MHz	87,7 MHz

Tabulka 1: Využití zdrojů obvodu STRATIX I v závislosti na použitém nástroji.

Nástroj:	DK(VHDL) + Synplify	Synplify + Quartus	DK(EDIF)	DK (VHDL) + Quartus
Log. zdroje	3160 ALUTs	4128 ALUTs	10392 LUTs	3155 ALUTs
Paměť	x	5376 bit	5376 bit	5376 bit
Počet registrů	2051	2054	1992	2031
Časování	160,2 MHz	143,74 MHz	63,5 MHz	131,7 MHz

Tabulka 2: Využití zdrojů obvodu STRATIX II v závislosti na použitém nástroji.

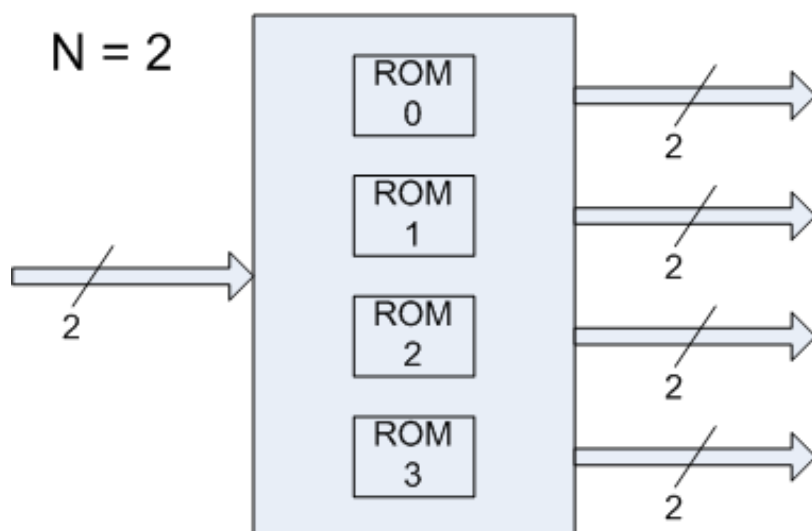
Nástroj:	DK(VHDL) + Synplify	Synplify + ISE	DK(EDIF)	DK (VHDL) + ISE
Log. zdroje	4028 LUTs	2464 Slices	9844 LUTs	3630 Slices
Paměť	2 Block RAM	2 Block RAM	36864	4 Block RAM
Počet registrů	2235	2235	1988	2037
Časování	175,5 MHz	105,99 MHz	43,68 MHz	79,7 MHz

Tabulka 3: Využití zdrojů obvodu VIRTEX v závislosti na použitém nástroji.

5.1 ToDo

Ve stávající implementaci popisované architektury tvoří kritickou cestu výpočet akumulované metriky v ACSU. Možnou optimalizací se nabízí přepracovat výpočet metriky jednotlivých hran grafu. Metrika hrany je vypočítána na základě ohodnocení hrany a vstupního symbolu (např. viz obrázek 4). Výpočet lze nahradit pamětí typu ROM. Paměť lze adresovat přímo vstupním symbolem dekodéru. Výstupem paměti jsou metriky pro možné ohodnocení hran grafu. Počet možných ohodnocení hran je omezen parametrem N , tedy počtem výstupních bitů příslušného konvolučního kodéru, na hodnotu 2^N . Na obrázku 14 je znázorněna možná realizace jednotky BMU pomocí paměti ROM. Pro každé možné ohodnocení hrany je na obrázku 14 znázorněna paměť ROM, jejíž výstup je adresován vstupním symbolem dekodéru.

BMU



Obrázek 14: Příklad možné realizace výpočtu metriky pomocí paměti ROM

Vstupní symbol / Ohodnocení hrany	0	1	2	3
0	2	1	1	0
1	1	2	0	1
2	1	0	2	1
3	0	1	1	2

Tabulka 4: Hamingova vzdálenost vstupního symbolu a ohodnocení hrany (N = 2)

Zdrojové kódy dekodéru jsou psány parametricky. Parametry určují velikosti polí a datové šířky hodnot signálů. V některých případech dochází k výběru hodnot signálů pomocí adresace v poli hodnot. Tato adresace se promítne do implementace zvýšením složitosti logiky obvodu. Pro efektivnější implementaci se nabízí možnost přepracovat dekodér pro konkrétní hodnoty parametrů vhodných pro určenou aplikaci.

5.1.1 Soft dekódování

Stávající implementace viterbi dekodéru počítá metriky průchodu grafem na základě tzv. "hard decoding". Jak již bylo zmíněno dříve je při "hard decoding" počítána metrika na základě Hamingovi vzdálenosti. Pro "soft decoding" jsou vstupy dekodéru reprezentovány vícebitovou hodnotou.

Euklidovská vzdálenost je počítána jako rozdíl čtverců dvou vektorů. Pro případ viterbi dekodéru je to příchozí symbol a ohodnocení hrany. Pomocí jednoduchých úprav lze tento výpočet převést na jednoduché sčítání, které vede k jednoduché implementaci v obvodu FPGA.

Mějme konvoluční kodér, který generuje 2-bitový výstup (00, 01, 10, 11). Bitová reprezentace (Return-to-Zero) lze převést do vektorové reprezentace (Non-Return-Zero), jak je uvedeno v tabulce 5.

Každý přijatý symbol je reprezentován jako vektor $v_r = (r_0, r_1)$, kde r_0 a r_1 určují význam (sílu) jednotlivých vstupních signálů dekodéru.

Výpočet Euklidovské vzdálenosti je následující:

$$D = ||v_r - v_i||^2 = ||v_r||^2 - 2(v_r * v_i) + ||v_i||^2 \quad (3)$$

Binární reprezentace	Vektorová reprezentace
00	1, 1
01	1, -1
10	-1, 1
11	-1, -1

Tabulka 5: Převod symbolů z binární do vektorové reprezentace

, kde v_i je vektor ohodnocení hran, tedy $v_i = (\pm 1, \pm 1)$. Z toho lze odvodit, že

$$\|v_i\|^2 = (\pm 1)^2 + (\pm 1)^2 = 2 \quad (4)$$

Jednoka ACS porovnává kódovou vzdálenost mezi přijatým symbolem a ohodnocením hrany. Do každého bodu mřížky vedou právě dvě hrany z předešlých stavů mřížky. Předešlé stavy jsou ohodnoceny akumulovanou metrikou (ohodnocení nejlepší cesty vedoucí do tohoto stavu). Hodnota akumulované metriky aktuálního stavu je vypočítána na základě akumulovaných metrik předešlých stavů a metrik aktuálních dvou hran. Výsledná metrika stavu je dána součtem nejlepší akumulované metriky z předchozích dvou stavů a nejlépe ohodnocenou aktuální hranou. Označme ohodnocení hran vedoucí do jednoho stavu mřížky vektory $v_i^{(0)}$ a $v_i^{(1)}$.

Vzdálenosti těchto vektorů od vstupního symbolu je vypočítána následovně:

$$D_0 = \|v_r\|^2 - 2(v_r * v_i^{(0)}) + \|v_i^{(0)}\|^2 \quad D_1 = \|v_r\|^2 - 2(v_r * v_i^{(1)}) + \|v_i^{(1)}\|^2 \quad (5)$$

Vypočtené hodnoty porovnáme a lepší ohodnocení přičteme k vybrané akumulované metrice předchozího stavu.

Porovnání a výpočet hodnot ze vztahu 5 můžeme zjednodušit vzhledem ke vztahu 4 a faktu, že v_r je v obou výrazech totožný, na výběr minimálních hodnot dvou skalárních součinů.

$$\min((-2(v_r * v_i^{(0)})), (-2(v_r * v_i^{(1)}))) = \max((v_r * v_i^{(0)}), (v_r * v_i^{(1)})) \quad (6)$$

Vztah 6 lze dále rozepsat:

$$\max((\{r_0, r_1\} * \{\pm 1, \pm 1\}), (\{r_0, r_1\} * \{\pm 1, \pm 1\})) = \max(\pm r_0 \pm r_1, \pm r_0 \pm r_1) \quad (7)$$

, kde znaménka ve výrazu závisí na aktuálních hodnotách vektorů $v_i^{(0)}$ a $v_i^{(1)}$.

Z výše popsaných úprav můžeme výpočet metriky hran zjednodušit na jednoduché operace sčítání či odčítání.

5.1.2 Puncturing

Metoda takzvaného děrování (puncturing) je používána pro zvýšení informačního poměru zakódovaných dat. Při serializaci zakódovaných dat konvolučního kodéru jsou dle předem stanoveného schématu vynechávány výstupy kodéru. Při dekodování "děrovaného" kódu se musí bitová posloupnost opět převést do posloupnosti N-bitových symbolů. Symboly musí být doplněny o vynechané bity podle stejného schématu, jakým byly vynechávány. Bity jsou nahrazeny například nulovou hodnotou, jelikož neznáme jejich původní hodnotu. Nahrazené bity nejsou brány v úvahu při výpočtu metrik ve Viterbi dekodéru.

5.1.3 Snížení latence výstupu

Pro snížení latence výstupu je možné z dekodéru vyvést výstup s bity v neuspořádaném pořadí, jak bylo popsáno v kapitole 3.2.3 a zachytávat je pomocí paměti FIFO. Paměť FIFO můžeme zrealizovat tak, aby bylo možné po jeho naplnění přecházet dekodované bity paralelně ve správném pořadí. Paměť FIFO musí být dlouhá jako počet kroků zpětného průchodu.

6 Ověření funkce

Ověření dekodéru a kodéru je možné nahráním příslušného bitstreamu (.sof), který se nahraje do je jedné z vývojových desek:

- Nios Development Board, Cyclone Edition (osazena obvodem Altera Cyclone EP1C20F400C7)
- Nios Development Board, Stratix Edition (osazena obvodem Altera Stratix EP1S10F780C6)
- Stratix II EP2S180 DSP Development Board (osazena obvodem Altera Stratix II EP2S180F1020C3)

Vstupní data pro kodér/dekodér jsou předávána pomocí ethernetového rozhraní za pomoci UDP protokolu. Po nahrání příslušného bitstreamu má vývojová deska nastavenou MAC adresu na 00:07:ed:0a:05:f3 a přijímá na portu číslo 2000.

Velikost dat zasílaných pro kodér/dekodér je omezena velikostí jednoho ethernetového rámce. Pro kodér je tato velikost omezena na 636 bytů, protože po zakódování bude vysláno dvojnásobek bytů (1272 bytů + příslušné hlavičky protokolů odpovídají maximální délce jednoho ethernetového rámce). Data by měla být zakončena bytem nulové hodnoty, aby došlo ke korektnímu zakódování.

U dekodéru je situace opačná a může přijmout až 1272 bytů.

Náhodná data zakončená nulovým bytem je možné vygenerovat pomocí programu

`\gendata\Release\gendata.exe`.

Parametry kodéru a dekodéru jsou následující:

$$N = 1 \quad (8)$$

$$M = 7 \quad (9)$$

$$Poly0 = x^6 + x^5 + x^4x^3 + 1 \quad (10)$$

$$Poly1 = x^6 + x^4 + x^3 + x + 1 \quad (11)$$

$$ACS = 4 \quad (12)$$

$$TLENGTH = 42 \quad (13)$$

$$(14)$$

6.1 Ověření funkčnosti dekodéru

Funkci dekodéru lze ověřit nahráním bitstreamu do příslušného obvodu vývojové desky.

Bitstreamy pro jednotlivé vývojové desky jsou uloženy v souborech uvedených v tabulce 6.

Cílová vývojová deska:	Cesta:
Nios Development Board, Cyclone Edition	<code>\pb_eth_viterbi\pblaze_eth\CYCLONE_EDIF\pblaze_eth.sof</code>
Nios Development Board, Stratix Edition	<code>\pb_eth_viterbi\pblaze_eth\STRATIX_EDIF\pblaze_eth.sof</code>
Stratix II EP2S180, DSP Development Board	<code>\pb_eth_viterbi\pblaze_eth\STRATIXII_EDIF\pblaze_eth.sof</code>

Tabulka 6: Cesty k bitstreamům dekodéru pro jednotlivé vývojové desky.

Tyto bitstreamy lze do vývojové desky nahrát pomocí downloaderu vývojového prostředí Quartus II (viz obrázek 15).

Po nahrání bitstreamu vývojová deska čeká na příchozí ethernetový rámec obsahující data a po přijetí zobrazí na sedmi-segmentovém displeji počet přijatých bytů, zpracuje data a vyšle je přes ethernetové rozhraní. Rámec musí být ve formátu UDP a číslo cílového portu rovno 2000. Vývojová deska nemá přiřazenu žádnou IP adresu a nepodporuje protokol ARP. Pro komunikaci s vývojovou deskou je nutné přidat statický záznam v ARP tabulce. IP adresu zvolíme libovolnou a MAC adresa je 00:07:ed:0a:05:f3. V prostředí MS Windows vytvoříme statický záznam příkazem `arp -s XX.XX.XX.XX 00-07-ed-0a-05-f3`, kde XX.XX.XX.XX je libovolná zvolená adresa (viz obrázek 16).

Data pro dekodér lze zaslat pomocí jednoduchého programu uloženého ve spustitelném souboru `\sendudp\Release\sendup.exe`. Užití programu je následující:

```
sendup.exe <cílová IP adresa> <cílový port> [Zdrojová IP adresa] <vstupní soubor>
               <výstupní soubor>
```

Cílová IP adresa je zvolená IP adresa v záznamu ARP tabulky, cílový port je v našem případě roven 2000, zdrojová IP adresa je nepovinný parametr (pokud není zadána je nastavena automaticky), vstupní soubor obsahuje binární data se vstupními řetězci a do výstupního souboru se uloží přijatá data (použití programu viz obrázek 17).

Soubor se vstupními daty obsahuje zakódovaná data (případně s vnesenou chybou) pomocí konvolučního kodéru. Soubor se zakódovanými daty můžeme získat například pomocí ověření funkce konvolučního kodéru popsané v kapitole 6.2, kdy pro zakódování můžeme použít jakákoliv data. Pro ověření schopnosti dekodéru opravit vnesenou chybu editujeme binární soubor se zakódovanými daty ručně.

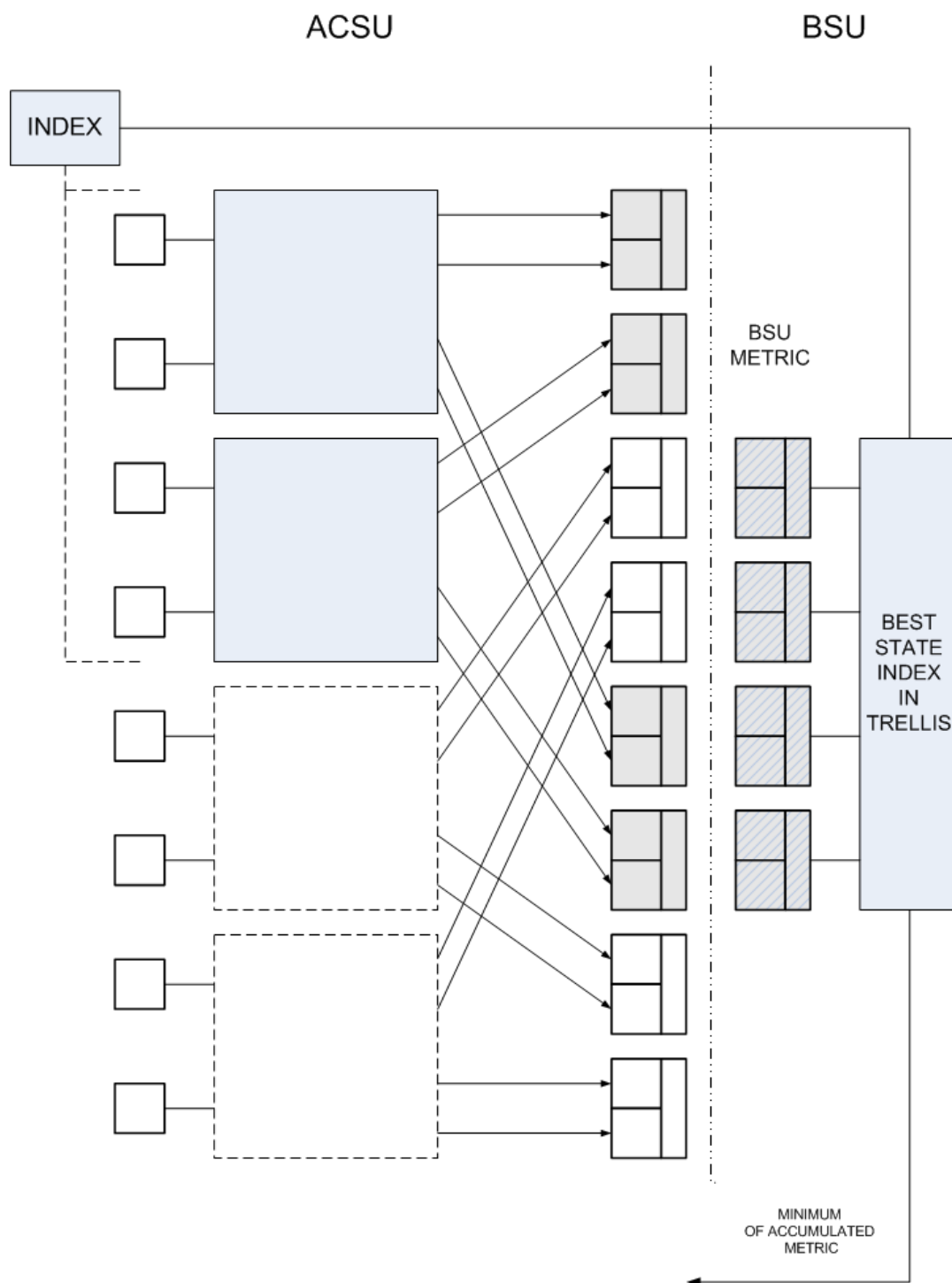
6.2 Ověření funkčnosti kodéru

Kodér lze ověřit stejným způsobem jako dekodér, kdy pomocí programu `sendudp.exe` použijeme jako vstupní soubor jakákoliv data. Přijatá data můžeme opět dekodovat pomocí dekodéru.

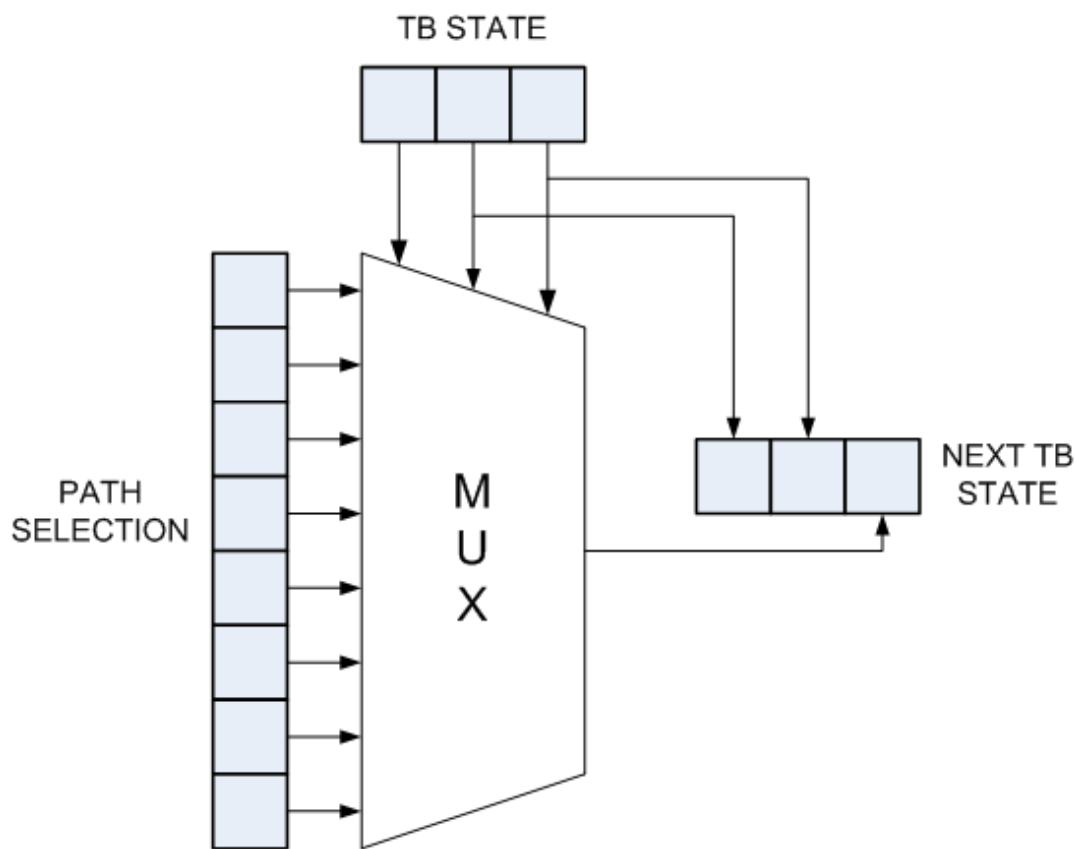
Bitstramy pro jednotlivé vývojové desky jsou uloženy v souborech uvedených v tabulce 7.

Cílová vývojová deska:	Cesta:
Nios Development Board, Cyclone Edition	<code>\pb_eth_conv\pblaze_eth\CYCLONE_EDIF\pblaze_eth.sof</code>
Nios Development Board, Stratix Edition	<code>\pb_eth_conv\pblaze_eth\STRATIX_EDIF\pblaze_eth.sof</code>
Stratix II EP2S180, DSP Development Board	<code>\pb_eth_conv\pblaze_eth\STRATIXII_EDIF\pblaze_eth.sof</code>

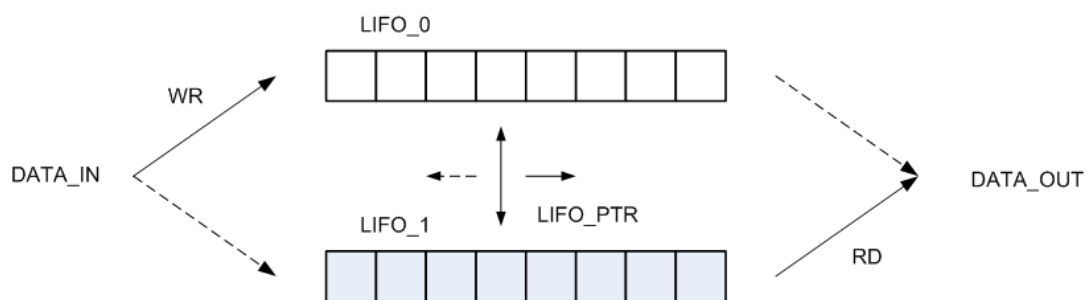
Tabulka 7: Cesty k bitstreamům kodéru pro jednotlivé vývojové desky.



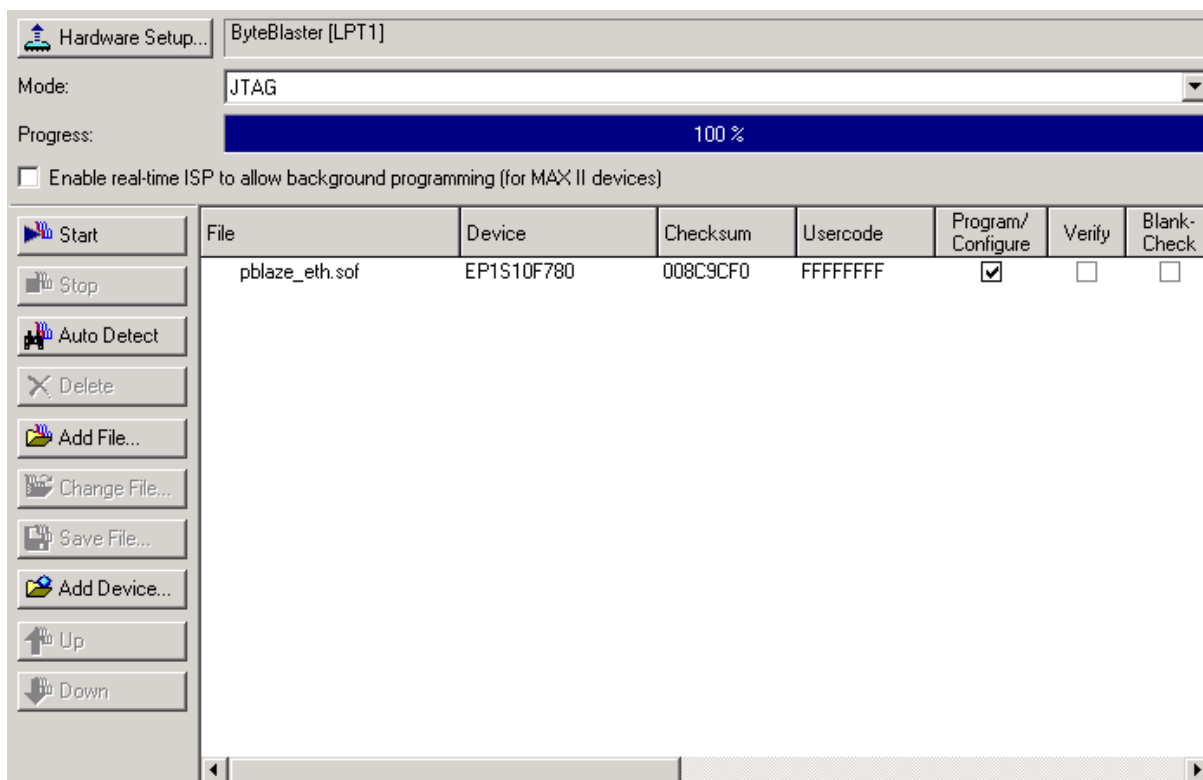
Obrázek 6: Struktura hybridní realizace jednotky ACS



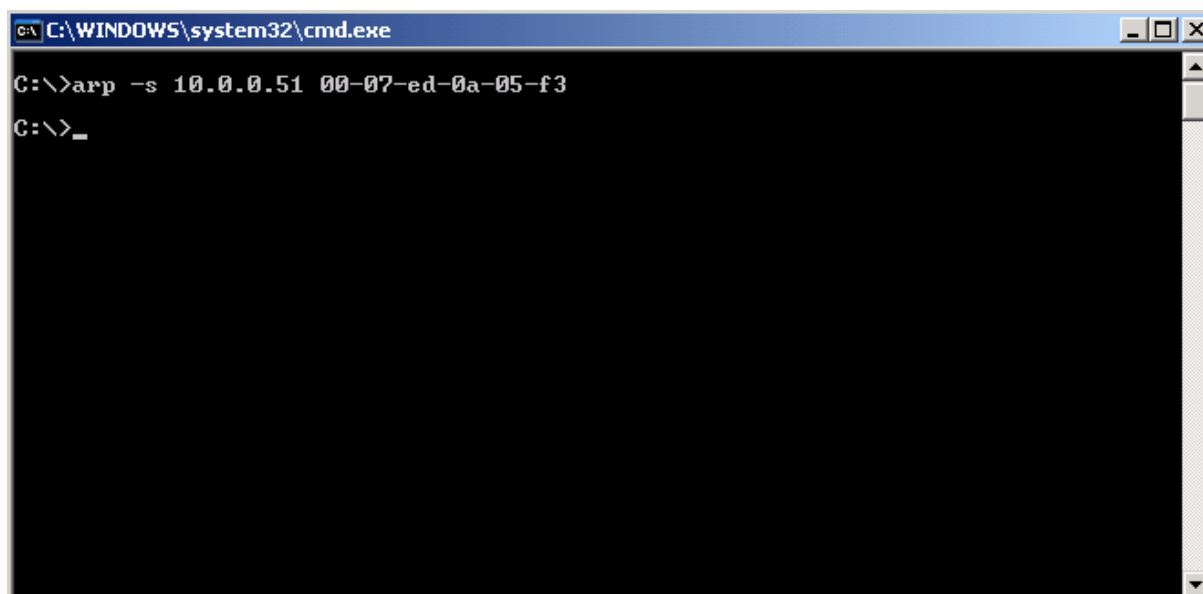
Obrázek 10: Logika pro výpočet dalšího stavu při zpětném průchodu grafem



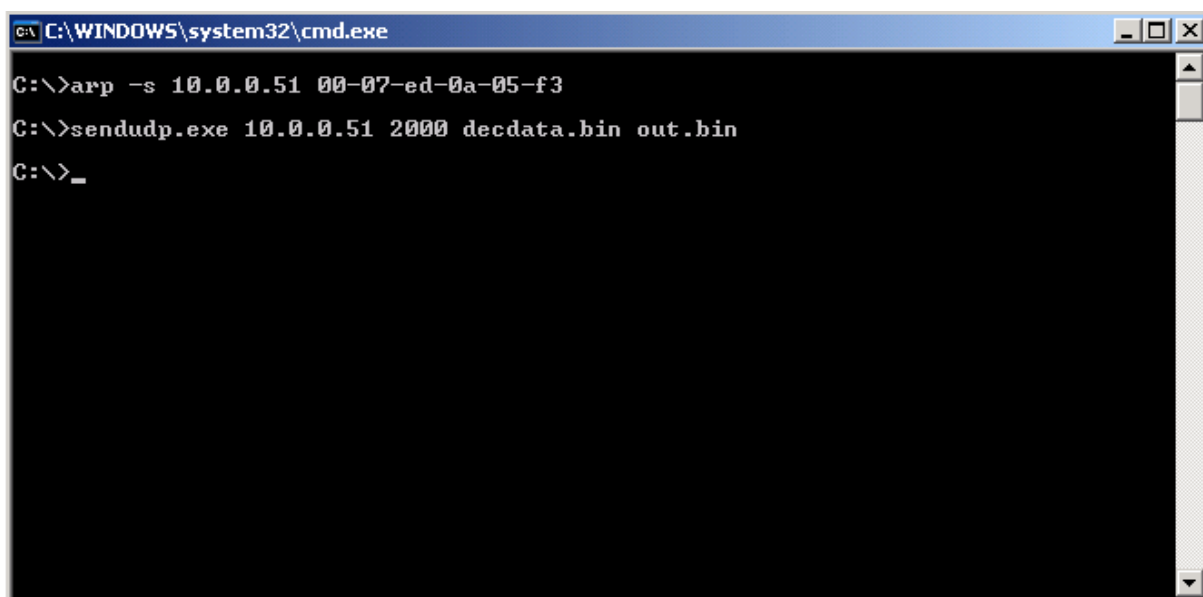
Obrázek 11: Princip záměny pořadí výstupních bitů pomocí pamětí LIFO



Obrázek 15: Dialogové okno downloaderu vývojového prostředí Quartus II.



Obrázek 16: Vytvoření statického záznamu ARP tabulky v prostředí MS Windows.



```
C:\WINDOWS\system32\cmd.exe  
C:\>arp -s 10.0.0.51 00-07-ed-0a-05-f3  
C:\>sendudp.exe 10.0.0.51 2000 decdata.bin out.bin  
C:\>_
```

Obrázek 17: Příklad použití programu pro odesílání dat.

7 Výpis obsahu CD-ROM

Na CD se nachází text dokumentu, zdrojové kódy konvolučního kodéru a Viterbi dekodéru, skript pro Matlab generující parametry zdrojových kódů, program pro zasílání UDP rámců a soubory pro naprogramování FPGA.

Příložené CD má následující adresářovou strukturu:

```
.
|-- doc/                text dokumentu ve formátu PDF
|-- gendata/            zdrojové kódy programu pro generování náhodných dat
|   '--Release          spustitelný program
|-- matlab/            skript pro vygenerování makro procedur a parametrů
|                       zdrojových kódů
|-- pb_eth_conv/       ověření funkce konvolučního kodéru
|   '-- pblaze_eth
|       |-- CYCLONE_EDIF bitstream pro ověření kodéru pro obvod Cyclone
|       |-- STRATIX_EDIF bitstream pro ověření kodéru pro obvod Stratix
|       '-- STRATIXII_EDIF bitstream pro ověření kodéru pro obvod Stratix II
|-- pb_eth_viterbhi/   ověření funkce Viterbi dekodéru
|   '-- pblaze_eth
|       |-- CYCLONE_EDIF bitstream pro ověření dekodéru pro obvod Cyclone
|       |-- STRATIX_EDIF bitstream pro ověření dekodéru pro obvod Stratix
|       '-- STRATIXII_EDIF bitstream pro ověření dekodéru pro obvod Stratix II
|-- sendudp/           program a jeho zdrojové kódy pro posílání UDP rámců
|   '--Release          spustitelný program
|-- viterbi/           zdrojové kódy Viterbi dekodéru a konvolučního kodéru
|   |-- standalone/    komponenta Viterbi dekodéru
|   |   |-- CYCLONE     zdrojové kódy ve VHDL pro obvod Cyclone
|   |   |   '--rev_1    syntetizované kódy (VQM)
|   |   |-- STRATIX     zdrojové kódy ve VHDL pro obvod Stratix
|   |   |   '--rev_1    syntetizované kódy (VQM)
|   |   |-- STRATIXII   zdrojové kódy ve VHDL pro obvod Stratix II
|   |   |   '--rev_2    syntetizované kódy (VQM)
|   |   |-- VIRTEX      zdrojové kódy ve VHDL pro obvod Virtex-4
|   |   |   '--rev_3    syntetizované kódy (EDF)
|   |   '-- test/       Zdrojové kódy pro simulaci kodéru a dekodéru
|-- README
```

Reference

- [1] Rizwan Rasheed. *Reconfigurable Viterbi Decoder for Mobile Platform* [online]. Dostupné na WWW: <http://www.ctr.kcl.ac.uk/mwcn2005/Paper/C200540.pdf>
- [2] Xilinx. *Viterbi Decoder v6.0* [online]. Dostupné na WWW: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/viterbi.pdf
- [3] Chip Fleming. *Tutorial on Convolutional Coding with Viterbi Decoding* [online]. Dostupné na WWW: <http://home.netcom.com/~chip.f/Viterbi.html>
- [4] *The Error Correcting Codes (ECC) Page* [online]. Dostupné na WWW: <http://www.eccpage.com/>
- [5] Russell Tessier, Sriram Swaminathan, Ramaswamy Ramaswamy, Dennis Goeckel and Wayne Burleson. *A Reconfigurable, Power-Efficient Adaptive Viterbi Decoder* [online]. Dostupné na WWW: [url](#)
- [6] Texas Instruments. *Viterbi Decoding Techniques for the TMS320C54x DSP Generation* [online]. Dostupné na WWW: <http://focus.ti.com/lit/an/spra071a/spra071a.pdf>
- [7] Altera. *Viterbi Compiler, User Guide* [online]. Dostupné na WWW: http://www.altera.com/literature/ug/ug_viterbi-compiler.pdf
- [8] L. Brackenbury, M. Cumpstey, S. Furber, P. Riocreux. *An Asynchronous Viterbi Decoder* [online]. Dostupné na WWW: <http://intranet.cs.man.ac.uk/apt/projects/lowpower/prest/async.vit.pdf>