VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

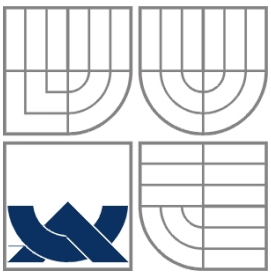
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PŘEHRÁVAČ MP3 SOUBORŮ V FPGA

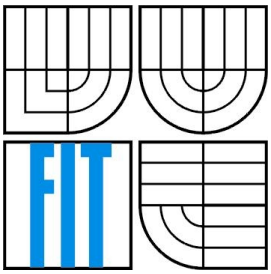
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. TOMÁŠ NÁPLAVA
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PŘEHRÁVAČ MP3 SOUBORŮ V FPGA

FPGA-BASED MP3 PLAYER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. TOMÁŠ NÁPLAVA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAN KAŠTIL

BRNO 2012

Abstrakt

Tato práce se zabývá návrhem a implementací hardwarové jednotky, která je schopná přehrávat soubory formátu MPEG-1 layer III, dle standardu ISO/IEC 11172-3. Jsou uvedeny výhody použití formátu MP3 a principy, díky kterým je možné komprimovat velikost výsledných hudebních nahrávek. Dále je důkladně prostudován formát souboru, všechny části hlavičky a způsob zakódování informace. Dekódování dat je rozděleno do několika po sobě jdoucích, víceméně samostatných funkčních jednotek a tyto jednotky jsou navrženy a popsány v jazyku pro popis obvodů VHDL. Dále je diskutována problematika čipů FPGA – programovatelných hradlových polí. Těch je využito k fyzické realizaci MP3 přehrávače. Je vybrána vývojová deska s takovým FPGA čipem a dalšími prostředky, které umožní syntézu celého obvodu a přehrávání v reálném čase.

Abstract

This work deals with the design and implementation of a hardware unit that is capable of playing MPEG-1 Layer III files, compliant with ISO/IEC 11172-3. There are given the benefits of using the MP3 format and principles that make it possible to compress the size of the resulting music recordings. The file format and all parts of the header are thoroughly studied as well as the method of encoding information. The process of the data decoding is divided into several consecutive, more or less discrete functional units and these units are designed and described in a hardware description language VHDL. There are also discussed features of FPGA chips - programmable gate arrays. Those are used for physical realization of the MP3 player. A development board is selected, including such an FPGA chip and other resources that allow synthesis of the entire circuit and playback in real time.

Klíčová slova

formát souboru MP3, MP3 přehrávač, MP3 dekodér, konfigurovatelný HW, FPGA, VHDL

Keywords

MP3 file format, MP3 player, MP3 decoder, configurable HW, FPGA, VHDL

Citace

Tomáš Náplava: Přehrávač MP3 souborů v FPGA, diplomová práce, Brno, FIT VUT v Brně, 2012

Přehrávač MP3 souborů v FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kaštila. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Náplava

23.5.2012

Poděkování

Chtěl bych tímto poděkovat Ing. Janu Kaštilovi za pomoc s řešením problémů během práce na projektu.

Obsah

1 Úvod

Sluch hraje v našem životě důležitou roli v poznávání okolního světa. Není sice tak důležitý jako zrak, který nám dodává cca 90% všech informací o dění okolo nás, ale jistě má i své nesporné výhody. Sluchové vjemy můžeme vnímat paralelně se zrakovými, což může být užitečné při učení – nově věci se snáze zapamatují, pokud jsou vnímány více smysly. Další důležitou vlastností zvukem přenesené informace je to, že posluchač se nemusí soustředit či jakkoliv připravovat, aby vjem zaznamenal, to je vhodné pro signály, upozorňující na nějakou událost. V neposlední řadě můžeme pohlížet na sluch jako nezbytnou součást pro mezilidskou komunikaci řečí, či jako nástroj pro estetické vnímání při poslechu hudby.

Za posledních 15 let jsme mohli zažít boom digitálních audio nahrávek ve formátu MP3, zatímco nahrávky na kompaktních nosičích ustupují do pozadí. Tuto změnu lze pozorovat i z postoje nahrávacích společností, které zprvu braly převody originálních CD do formátu MP3 a jejich následné šíření po internetu spíše jako pirátství, než jako převratnou novinku. Nyní se snaží samy nabídnout spotřebitelům nahrávky ve formátu MP3. I hudební skupiny se musí s tímto fenoménem smířit a nemohou očekávat takové zisky z prodeje CD, jako tomu bylo v minulosti.

Důvodů pro oblíbenost tohoto formátu je jistě mnoho, hlavní z nich je rapidní zmenšení velikosti souboru oproti uložení pomocí klasických 16-bitových PCM vzorků, tak jak je známe z kompaktních disků, na cca jednu desetinu. To je velice vhodné jak pro šetření místem při ukládání na pevný disk, tak i pro přenášení dat po síti. Navíc je rozdíl v kvalitě mezi originální nahrávkou a MP3 pro většinu lidí nepostřehnutelný, přičemž se dá kvalita MP3 ovlivnit volbou datového toku a vzorkovací frekvence. K dispozici jsou volně stažitelné programy pro přehrávání i pro zakódování do formátu MP3.

Cílem této práce je prostudovat formát souborů MP3 a navrhnout podle normy MP3 dekodér schopný přehrávat tyto soubory. Jako důkaz proveditelnosti bude MP3 dekodér hardwarově implementován v cílové technologii FPGA. Ve druhé kapitole je podrobně rozebrán formát souboru MP3. Třetí kapitola nabízí obecný pohled na MP3 enkodér. Ve čtvrté kapitole je vysvětlena funkce částí dekodéru a související výpočty. V páté kapitole se k krátkosti představí technologie FPGA a nejpoužívanější nástroje pro návrh, simulaci a syntézu. Šestá a sedmá kapitola pojednává o návrhu a implementaci částí dekodéru ze 4. kapitoly. Na závěr jsou zhodnoceny výsledky celé práce.

2 Formát souboru MP3

2.1 Obecná charakteristika

Formát MP3 je specifický digitální formát, určený pro ukládání a kompresi audio nahrávek. Byl vyvinut výzkumnou skupinou zvanou MPEG (Motion Picture Experts Group) a v roce 1993 ustanoven jako standard organizací ISO pod označením ISO/IEC 11172-3, nebo také MPEG-1 Audio Part 3 [1].

Soubory MP3 využívají ztrátovou kompresi, což znamená, že dekomprimováním zakódovaného souboru už nelze získat zpět přesnou originální zvukovou stopu. Ve většině aplikací je taková vlastnost nepřípustná, jako např. U spustitelných binárních souborů, kde by překlopení jediného bitu na nesprávnou hodnotu mohlo způsobit havárii celého programu, avšak pro použití v audio technice je situace poněkud odlišná.

Nikoliv jako negativní vlastnost, ale v tomto případě spíše jako výhodu lze chápat určitou nedokonalost lidského sluchu. Ten není například schopen rozeznat příliš slabé zvuky, signály mimo slyšitelné frekvenční pásmo apod. Na tom je založena perceptivní komprese, kdy se data, která posluchač neslyší nebo není schopen vnímat, odstraní, protože ve výsledku nemají žádný vliv, a tím se ušetří místo.

2.2 Psychoakustický model

K určení, která data se mohou bez relativní újmy na kvalitě výsledku zanedbat, slouží psychoakustický model. To je matematický model, který se snaží co nejvěrněji popsat vnímání lidského ucha. Samotná norma obsahuje jeden příklad psychoakustického modelu, těch existuje samozřejmě více, a je na návrháři kodéru, který z nich použije pro dosažení co možná nejlepšího výsledku. Modely používané pro ztrátovou kompresi zvuku, využívají těchto čtyřech jevů, způsobených fyzikálními a anatomickými omezeními lidského sluchu [2]:

- frekvenční rozsah – člověk je schopen vnímat jen frekvence v pásmu 20Hz – 20kHz, záleží i na schopnostech jedince a na jeho stáří
- práh slyšitelnosti – je udáván v jednotkách dB, je to nejmenší možná hladina akustického tlaku, kterou je ucho ještě schopno zachytit. Je závislý na frekvenci signálu, pro signály o různé frekvenci se mohou prahy slyšitelnosti obecně lišit.
- simultánní maskování – vzniká, když je slabší signál přehlušen jiným, hlasitějším signálem. Tento signál způsobí dynamické zvýšení prahu slyšitelnosti slabšího signálu. Příkladem je obtížná konverzace na hlučné diskotéce.
- temporální maskování – toto maskování se objevuje před a po simultánním maskování. Je způsobeno jakousi setrvačností ve vnímání zvuku. Maskovaný signál může být ještě několik desítek milisekund neslyšitelný, i když už druhý, hlasitější signál odezněl.

2.3 Struktura souboru

Základními bloky, do kterých je každý soubor MP3 rozdělen, jsou rámce. Ty se skládají z hlavičky a samotných užitečných dat. Celý soubor je pak sérií takovýchto, po sobě jdoucích rámců. Kromě rámců nesoucích zvuková data se v souboru mohou objevit i metadata, popisující název a interpreta

nahrávky, rok vzniku, album, žánr a mohou dokonce obsahovat i obrázek přebalu alba. Tyto informace se zpravidla nacházejí na úplném začátku souboru a bývají uloženy ve formátu tzv. ID3 tagu verze 1 nebo 2. Jelikož se ID3 tag objevil až dodatečně po vzniku MP3, není součástí standardu a záleží tedy na návrháři dekodéru, jak s touto informací naloží. Nejjednodušší cestou je celý tag ignorovat, protože na dekódování souboru nemá žádný vliv. Existují však softwarové přehrávače, jako např. Winamp od společnosti Nullsoft, které dokážou tyto informace zobrazit, upravovat a dokonce dodatečně přidat.

2.3.1 Hlavička

Rámec se skládá z pěti částí: hlavičky, kontrolního součtu CRC, sideinfo, audio dat a dodatečných dat, přičemž povinně musí být přítomna pouze první, třetí a čtvrtá část. Rámec začíná hlavičkou dlouhou vždy 4B. Jednotlivé části hlavičky a jejich délky jsou znázorněny následující VHDL konstrukcí.

```
type MP3header is record
     syncword:             std_logic_vector(11 downto 0);
     ID:                   std_logic;
     layer:                std_logic_vector(1 downto 0);
     protection_bit:       std_logic;
     bitrate_index:        std_logic_vector(3 downto 0);
     sampling_frequency:   std_logic_vector(1 downto 0);
     padding_bit:          std_logic;
     private_bit:          std_logic;
     mode:                 std_logic_vector(1 downto 0);
     mode_extension:       std_logic_vector(1 downto 0);
     copyright:            std_logic;
     original_home:        std_logic;
     emphasis:             std_logic_vector(1 downto 0);
end record;
```

Význam jednotlivých částí hlavičky [1]:

- **syncword** (12 bitů) – jedná se o bitový řetězec dvanácti jedniček, jehož smyslem je identifikovat začátek hlavičky a tedy i celého rámce
- **ID** (1 bit) – prakticky vždy je nastaveno do jedničky, označující MPEG audio, nula je rezervovaná hodnota
- **layer** (2 bity) – index, označující použitou vrstvu, pro MPEG layer III je tato hodnota nastavena jako „01“
- **protection bit** (1 bit) – tento příznak udává, zda se bezprostředně za hlavičkou nachází 2B kontrolního součtu pro detekci chyby, nastavený bit značí, že součet není přítomen. Jelikož se předchozí hodnoty hlavičky nemění, lze v hexadecimálním editoru pohledem objevit začátek hlavičky, jejímž prefixem je 0xFFFB nebo 0xFFFA, v závislosti na tom, zda je nastavený protection bit.

- **bitrate index** (4 bity) – index datového toku, čím je tok vyšší, tím kvalitnější je záznam. V tabulce je uvedeno 15 možností, rozhoduje i to, jaká vrstva je použita. Dva po sobě jdoucí rámce nemusejí sdílet tutéž hodnotu, čímž se dosáhne proměnlivého bitrate.
- **sampling frequency** (2 bity) – vzorkovací frekvence vstupního signálu, 32, 44.1 nebo 48kHz
- **padding bit** (1 bit) – příznak, který je nastaven, pokud je použit 1B pro zarovnání dat kvůli dosažení zadaného bitrate. To je způsobeno nesoudělností počtu vzorků a vzorkovací frekvence, nastává pouze pro frekvenci 44.1 kHz.
- **private bit** (1 bit) – nemá určen žádný specifický význam
- **mode** (2 bity) – značí jestli je použit jeden nebo více kanálů, popř. konkrétní stereo mód
- **mode extension** (2 bity) – další nastavení pro mód joint stereo
- **copyright** (1 bit) – nastavený bit znamená, že soubor je chráněn autorskými právy
- **original/home** (1 bit) – pro kopii je tento bit vynulován, originální soubor ho má nastaven
- **emphasis** (2 bity) – doporučený způsob odstranění šumu, nepoužívá se

Po hlavičce následují nepovinné 2B pro ochranu dat, kontrolní součet slouží pro detekci chyby, jedná se o metodu CRC-16 s tímto generujícím polynomem G(X) = X16 + X15 + X2 + 1. Tímto kontrolním kódem je zabezpečena druhá polovina hlavičky a počet bitů použitých pro uložení vzorků. Samotný obsah vzorků není zabezpečen. Norma obsahuje v dodatcích algoritmus pro spočítání kontrolního součtu, pokud se tato hodnota liší od předpokládané, nastala chyba v přenosu. V takovém případě se doporučuje ztišit výstupní vzorky aktuálního rámce na minimum, nebo reprodukovat vzorky předchozího rámce.

2.3.2 Side info

Samotná data z hlavičky pro dekódování vzorků nestačí, a proto musí MP3 obsahovat další metadata, která se nazývají side info. Přicházejí hned po kontrolním součtu, pokud je použit. Délka těchto dat může nabývat hodnot 17 nebo 32B, v závislosti na tom, jestli se jedná o mono nebo stereo nahrávku (o tom informuje část hlavičky zvaná mode).

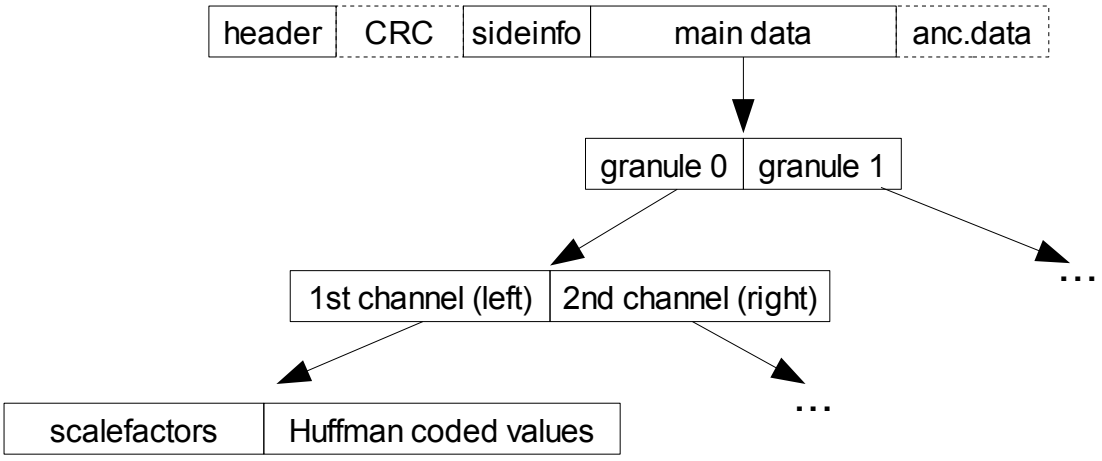
Význam částí side info:

- **main data beg** (9 bitů) – označuje negativní offset v bytech začátku audio dat od hlavičky aktuálního rámce – ukazuje na začátek logického rámce. Data aktuálního logického rámce mohou být totiž uložena i v předchozích rámcích, bude vysvětleno dále.
- **private bits** (3/5 bitů) – tyto bity nemají určen žádný zvláštní význam, lze je chápat jako nutné zarovnání celého side info na celé byty.
- **scfsi** (4/8 bitů) – neboli scale factor selector information je řetězec 4 bitů pro každý kanál, který určuje, zda se scalefactory rozdělené do čtyřech pásem přenášejí v každé granuli zvlášť, nebo jsou mezi oběma granulemi sdíleny. Pokud má bit hodnotu 0, daná skupina scalefactorů je různá pro obě granule. Scalefactory se uplatňují při dekódování ve fázi rekvantizace.
- **part 2,3 length** (12/24 bitů) – označuje v bitech celkovou velikost dat použitých pro uložení scalefactorů a audio vzorků v Huffmanově kódu. Díky této informaci lze spočítat, kde v souboru se nachází začátek hlavních dat pro každou granuli a začátek dodatečných dat.
- **big values** (9/18 bitů) – je to polovina počtu vzorků, spadajících do tzv. big values regionu.
- **global gain** (8/16 bitů) – tato hodnota je použita při dekódování pro rekvantizaci a vztahuje se na všechny vzorky napříč frekvenčním spektrem.
- **scalefac compress** (4/8 bitů) – udává, na kolika bitech jsou uloženy scalefactory. Podle typu okna jsou vzorky rozděleny do 12 nebo 21 pásem scalefactorů. Šestnáct možných variant z tabulky určuje délku v bitech zvlášť pro první a druhou polovinu skupiny scalefactorů.
- **block split flag** (1/2 bity) – signalizuje, že v bloku není použito klasické dlouhé okno. Pokud je v nule, je automaticky nastaven block type jako dlouhé okno.

- **block type** (2/4 bity) – tato položka se objevuje, jen pokud je nastaven block split flag nastaven. Určuje typ okna, dlouhé okno je rezervovaná hodnota, dalšími možnostmi jsou 3 krátká okna a startovní a koncové okno.
- **switch point** (1/2 bity) – tato položka se objevuje, jen pokud je nastaven block split flag. Jestliže je tento flag nastaven, je použito na tři nejspodnější pásma scalefactorů dlouhé okno a na zbytek vzorků okno dle nastavení block type. V opačném případě se aplikuje okno nastavené podle block type na všechny vzorky.
- **table select** (15/30 bitů) – jedná se o 3 pětibitové indexy pro každý kanál, které vybírají tabulky pro dekódování huffmanových bitů v regionech 0, 1 a 2. Použitím třech různých tabulek pro různé části spektra se dosáhne jisté adaptibility a tím i zlepšení komprese dat.
- **subblock gain** (9/18 bitů) – tato položka se objevuje, jen pokud je nastaven block split flag. Na rozdíl od global gain se uplatňuje pouze pro jednotlivá okna při rekvantizaci
- **region address1** (4/8 bitů) – tato hodnota označuje začátek regionu 1 v oblasti big values, lze ji interpretovat jako počet pásem scalefactorů z regionu 0, které je nutno přeskočit, abychom se dostali na začátek regionu 1. Například, pokud je region address 1 roven nule, vyplývá z toho, že region 0 je prázdný. Odečtením jedničky od této hodnoty získáme délku regionu 0, vyjádřenou v počtu pásem scalefactorů.
- **region address2** (3/6 bitů) – obdobná položka jako region address 1, odečtením jedničky od této hodnoty získáme délku regionu 1. Velikost regionu 2 lze získat na základě znalosti celkové velikosti big values a velikosti regionu 0 a 1.
- **preflag** (1/2 bity) – účelem toho příznaku je dodatečné zesílení vysokofrekvenčních kvantizovaných hodnot. Pokud je flag nastaven, přičte se z tabulky hodnota z příslušného pásma k scalefactoru. Pro krátké okna se nepoužívá.
- **scalefac scale** (1/2 bity) – scalefactory jsou logaritmicky kvantovány. Pokud je scalefac scale v jedničce, pak je kvantizační krok roven dvěma, jinak je roven odmocnině ze dvou.
- **count1 table select** (1/2 bity) – podobné jako položka table select, ale vybírá se pouze ze dvou huffmanových tabulek. Tabulky jsou určeny pro dekódování oblasti count1, která může obsahovat pouze 3 možné hodnoty: ±1 a 0.

2.3.3 Hlavní data

Část zvaná sideinfo je následována hlavními daty, nesoucími užitečnou informaci. Hlavní data obsahují z důvodu příznivějšího poměru mezi metadaty a užitečnými daty rovnou dvě granule. Každá granule obsahuje pro jeden kanál scalefactory (pokud jsou přítomny) a 576 vzorků. Protože jsou vždy přítomny 2 granule, při zpracování v jednokanálovém módu (mono) získáme 2x576=1152 vzorků, v případě zpracování dvoukanálového módu (stereo) obsahuje jeden rámec 2x2x576=2304 vzorků.



Obrázek 1: Struktura rámce

Délka zmíněného bloku 576 hodnot je variabilní, jelikož jsou vzorky zakódovány pomocí statického Huffmanova kódu s proměnlivou délkou. Jednotlivé vzorky pak představují amplitudy v daných frekvenčních pásmech, první vzorek začíná na frekvenci 0 Hz, poslední s indexem 576 odpovídá Nyquistově frekvenci, tedy polovině vzorkovací frekvence.

Z pohledu posouzení kvality poslechu se jeví některé frekvence kritičtější než ostatní, jelikož lidské ucho vnímá různé frekvence s různou odezvou. Zejména vzorky s nízkou frekvencí by měly nést co nejmenší chybu, zatímco malé zkreslení vzorků s nízkou amplitudou blížících se maximální frekvenci nemusí mít na výsledek tak dramatický vliv. Z tohoto důvodu je blok rozdělen na 5 oblastí, které se zakódují samostatně, s různou kvalitou.

Obrázek 2: Rozložení vzorků v granuli

Pří dekódování hodnot z bloku jsou první na řadě bigvalues, jsou to vzorky s nejnižší frekvencí a nejvyššími hodnotami amplitud, které se mohou pohybovat mezi hodnotami -8206 až +8206. Pro lepší efektivitu pří kódování je region bigvalues rozdělen do dalších třech podregionů, z nichž každý může používat jinou Huffmanovu tabulku. Počet vzorků z celého regionu je dvojnásobek hodnoty bigvalues, uvedené v části sideinfo. Je to proto, že při dekódování jednoho kódového slova dostáváme rovnou dvě hodnoty vzorků.

Po bloku hodnot bigvalues následuje region zvaný count1. V tomto pásmu nedosahují frekvenční vzorky tak vysokých amplitud jako u předchozích. Region je navržen tak, aby stačily pro uložení vzorků pouze 3 hodnoty, -1, 0 a +1. Vzhledem k tomuto zjednodušení jsou v tomto regionu k dispozici pouze dvě možné kódovací Huffmanovy tabulky oproti třiceti tabulkám v regionu bigvalues. Podobně jako v předchozím regionu i zde se dekódováním jednoho kódového slova získá více vzorků najednou, a to 4.

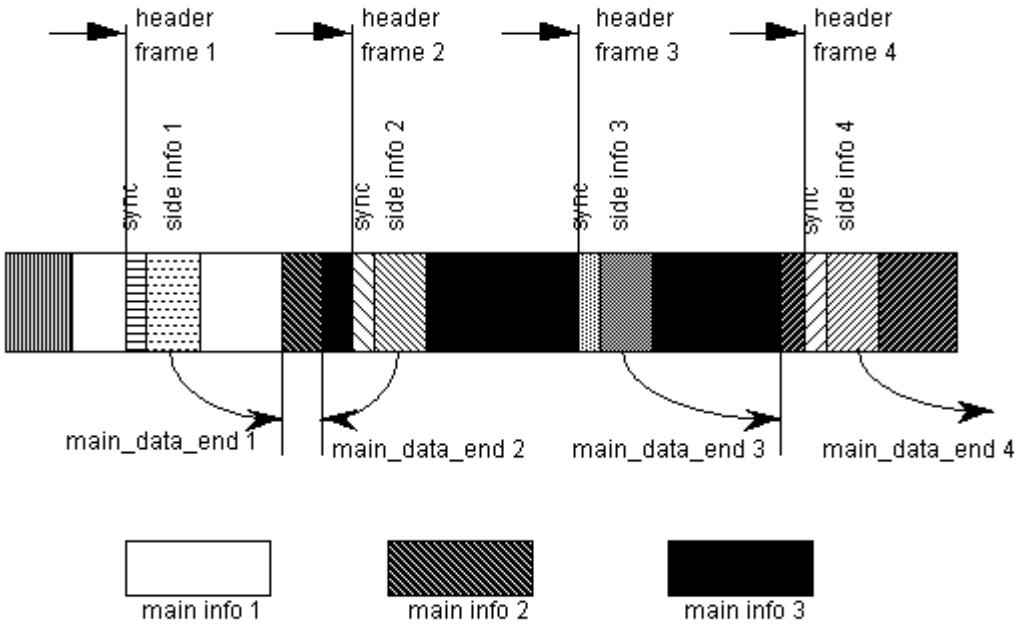
Na první pohled by se mohlo zdát zvláštní, že nemáme informaci o tom, jak velký, nebo kolik vzorků spadá do regionu count1. Ve skutečnosti však můžeme tento údaj zjistit odečtením hodnoty part 2,3 length od hodnoty bigvalues. Vzorky z regionu rzero totiž nezabírají ani jediný byte. Dekódování frekvenčních vzorků z bloku je ukončeno tehdy, když vyčerpáme všechny bity tohoto bloku, nebo získáme žádaných 576 vzorků, podle toho, která situace nastane dříve. V případě, že ještě není dekódováno všech 576 vzorků a žádné bity už nezbyly, jednoduše nastavíme zbytek vzorků na hodnotu 0. Tím vznikne poslední region rzero, který obsahuje jen samé nuly. Je jasné, že tuto informaci není potřeba do MP3 souboru ukládat a je doplněna automaticky při dekódování. Region rzero reflektuje velmi nízké amplitudy vysokých frekvencí, které se při kódování ořežou.

2.3.4 Fyzický a logický rámec

Jistou zvláštností formátu MP3 je, že rozlišuje pojmy fyzický a logický rámec. Logický rámec, který byl popisován v předchozím textu, je blok dat, který se dá dekódovat nezávisle na ostatních rámcích. Kvůli kódování s proměnnou délkou se může stát, že dva po sobě jdoucí rámce mají poměrně rozdílnou velikost, a proto je umožněno velkým logickým rámcům, aby byly rozprostřeny mezi více fyzických rámců. Situaci ilustruje obrázek [1].

Každý fyzický rámec začíná hlavičkou a blokem sideinfo. První rámec obsahuje kromě svých hlavních dat i hlavní data z rámce 2 a část rámce 3. Hlavní data rámce 3 jsou tak obsáhlá, že jsou

uložena po částech v prvních třech rámcích. V rámci 0 by se teoreticky data z třetího rámce nemohla objevit, protože je dovoleno použít maximálně dva předchozí fyzické rámce. Dalším omezením je to, že logický rámec nesmí přesáhnout hranici svého fyzického rámce. To je docela logický požadavek, protože jinak by dekodér musel vidět „do budoucnosti“. Při dekódování tedy postačuje znát poslední tři rámce, včetně aktuálního. O začátku logického rámce informuje část struktury sideinfo, zvaná main data beg.



Obrázek 3: Uložení hlavních dat mezi rámci [1]

3 Mp3 enkodér

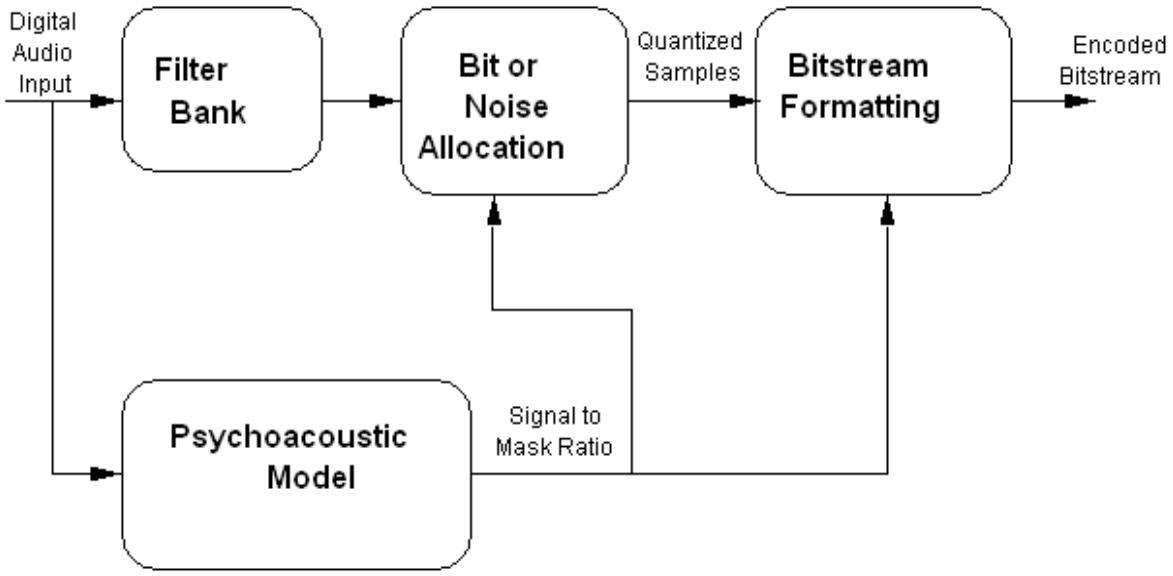
K zakódování toku digitálních audio dat do komprimované podoby MP3 slouží MP3 enkodér. Je vhodné, aby byl na vstupní vzorky aplikován hornopropustní filtr s mezní frekvencí 2 – 10Hz. Smyslem tohoto filtru je ořezání nízkofrekvenčních tónů a zmenšení datového toku v nejnižších frekvenčních pásmech, což se pozitivně projeví na kvalitě poslechu.

Vstupní signál je přiveden do bloků filter bank a do bloku, implementující psychoakustický model. Zde je provedena diskrétní Fourierova transformace a získá se frekvenční spektrum. Na základě vlastností spektra a na charakteristice psychoakustického modelu se určí prahové hodnoty pro maskování.

Blok filter bank je vlastně sadou 32 pásmových propustí. Celé frekvenční pásmo je rovnoměrně rozděleno těmito filtry, každý filtr propouští pouze část spektra. Výstupem jednoho filtru je 18 vzorků, což dá pro všech 32 filtrů celkem 576 vzorků, které se uloží v jedné granuli.

V posledním článku enkodéru bitstream formatting se poskládá celý fyzický rámec dohromady, vytvoří se hlavička a spočítá se kontrolní součet. Dále se do výstupního toku bitů uloží struktura sideinfo obsahující informace nutné k dekódování souboru. V této části enkodéru také probíhá komprese vzorků Huffmanovým kódem s proměnnou délkou.

Nutno podotknout, že žádný konkrétní algoritmus pro zakódování do formátu MP3 není standardizován. To byl od tvůrců MP3 úmyslný krok, čímž ponechávají prostor návrhářům při realizaci jejich enkodérů, a očekává se, že si sami navrhnou vhodné algoritmy. Proto je možné setkat se s MP3 enkodéry různých kvalit a rychlostí kódování. Naopak co je v normě detailně popsáno, je struktura zakódovaného souboru, význam jednotlivých složek a proces dekódování. Je zde řečeno, co se má udělat, ovšem nikoliv jak, chybí informace o tom, jak efektivně realizovat různé operace, či jak výpočet optimalizovat.

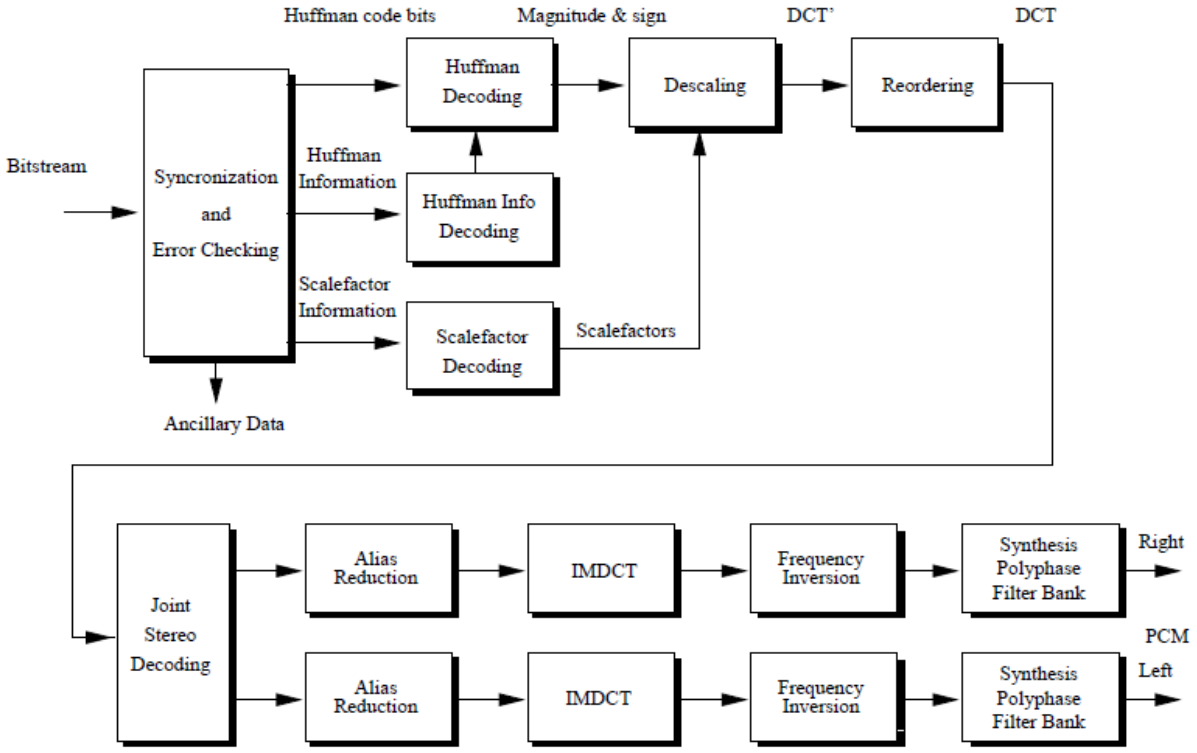


Obrázek 4: Struktura enkodéru [1]

4 Mp3 dekodér

Mp3 dekodér slouží pro dekódování vstupního bitového toku zpět na audio vzorky modulované pomocí PCM. To jsou vzorky amplitud snímané v pravidelných okamžicích a jejich kvalita tedy závisí na vzorkovací frekvenci a počtu bitů na jeden vzorek. V případě audio CD se setkáváme se vzorky o 16 bitech, což dává celkem 32768 možných hodnot v záporném a kladném pásmu. Na PCM vzorky se pak jednoduše aplikuje DA převodník, který s ohledem na vzorkovací frekvenci produkuje analogový signál, který se připojí ke sluchátkům nebo reproduktorům.

Dekodér plní opačnou funkci než enkodér, skládá se z navzájem víceméně nezávislých bloků, vykonávajících inverzní operaci k operacím odpovídajících bloků v enkodéru a logicky i v opačném pořadí. Na rozdíl od enkodéru se zde už nepracuje s psychoakustickým modelem, protože ve fázi dekódování už není potřeba, data lze jednoznačně dekódovat. Obrázek zachycuje více detailů než schéma enkodéru s ohledem na samostatnou implementaci jednotlivých bloků. V dalších podkapitolách uvedu funkci a výpočty v částech dekódovacího řetězce.



Obrázek 5: Části dekodéru [3]

4.1 Synchronizace

První částí dekodéru je blok, který má za úkol synchronizaci a dekódování částí rámce. První fyzický rámec, který se nemusí nutně objevit hned z kraje souboru, je identifikován začátkem hlavičky – sekvencí dvanácti jedničkových bitů. Jakmile je známa adresa hlavičky, je možné přečíst zbytek hlavičky a dále položky struktury sideinfo. Dalším problémem, který je v této části vyřešen, je

sestavení logického rámce, tedy nalezení hlavních dat k odpovídající hlavičce a části sideinfo. Délka fyzického rámce se vypočítá podle následujícího vzorce.

$$framesize = \frac{144 * bitrate}{sampling\ frequency} + padding\ bit \quad (1)$$

V případě vzorkovací frekvence 44.1 kHz se může stát, že výsledkem není celé číslo, pak se musí desetinná část výsledku oseknout, pro ostatní frekvence tato situace nenastává. Například pro vzorkovací frekvenci 44.1 kHz, bitrate 160 kbps a padding bit roven nule se délka rámce vypočítá takto:

$$framesize = \frac{144 * 160\,000}{44\,100} + 0 = 522\,B \approx 0x20A$$

Pokud by první fyzický rámec byl umístěný bezprostředně na začátku souboru, tedy na adrese 0x00, musel by druhý rámec začínat na adrese 0x20A, zde se musí objevit další sekvence dvanácti jedniček.

4.2 Huffmanův dekodér

Po fázi synchronizace a nalezení logických rámců se může přistoupit k dekódování frekvenčních vzorků komprimovaných statickým Huffmanovým kódem. Toto kódování je nejvýznamnějším faktorem, stojícím za vysokým kompresním poměrem souboru Mp3. Nyní si ho v krátkosti popíšeme.

4.2.1 Huffmanovo kódování

Huffmanovo kódování je entropické kódování s proměnnou délkou kódového slova. Komprese je dosaženo tím, že více pravděpodobným vstupním symbolům, u kterých se předpokládá, že se budou objevovat častěji, se přiřadí kratší kódy, zatímco zřídka se objevující symboly povedou na delší kódová slova. Optimální délka kódového slova pro symbol s se dá vypočítat jako $l = -\log_2(p(S))$ při použití binární abecedy, kde p(S) značí pravděpodobnost výskytu symbolu s na vstupu.

Jelikož je délka kódových slov různá v závislosti na pravděpodobnosti symbolů, narážíme na problém, jak určit, kde končí jedno kódové slovo a začíná druhé, když nejsou použity žádné speciální oddělovací znaky. Proto musí kódy s proměnlivou délkou splňovat tzv. prefixovou vlastnost, kdy pro všechny kódová slova musí platit, že nejsou prefixem žádného jiného kódového slova. Tato vlastnost zaručuje, že dekódování je jednoznačné, i když dopředu není známa délka aktuálního kódového slova.

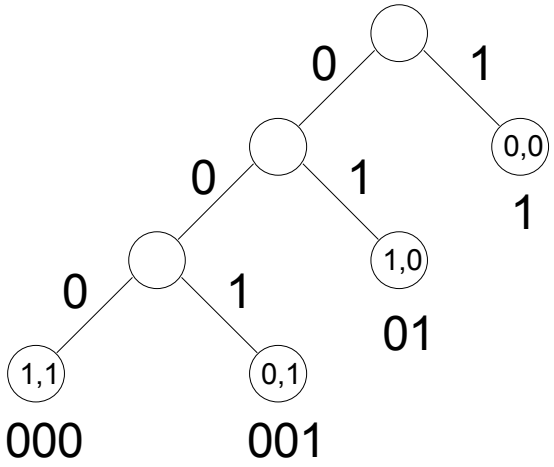
Kódování probíhá ve dvou fázích. V té první se shromáždí informace o pravděpodobnosti výskytu všech symbolů. Ve druhé fázi se začne budovat Huffmanův binární strom ve směru od listů ke kořeni, kde listy představují jednotlivé symboly. Vybere se dvojice uzlů s nejmenší pravděpodobností a k nim se vytvoří nový, rodičovský uzel, který reprezentuje nový symbol s pravděpodobností rovné součtu pravděpodobností obou synovských uzlů. Obě hrany z nového uzlu jsou ohodnoceny, jedna nulou a druhá jedničkou. Tento postup se opakuje až do té doby, kdy získáme poslední uzel – kořen stromu s pravděpodobností rovné jedné. Kódová slova jednotlivých symbolů jsou pak opačné sledy ohodnocení hran při průchodu stromem od příslušného listu ke kořeni.

Dekódování je poměrně jednoduchý proces, který pracuje s takto vytvořeným binárním stromem. V prvním kroku se označí kořen stromu jako aktuální uzel. Pokud je aktuální uzel uzlem listovým, pak je průchod stromem ukončen a na výstup se zapíše symbol uložený v tomto listovém uzlu. Pokud tomu tak není, přečte se ze vstupu jeden bit a podle jeho hodnoty se rozhodne, zda se má dále pokračovat v levém, nebo v pravém podstromu.

4.2.2 Huffmanovy tabulky

Přestože se během kódování nemůže statický Huffmanův strom měnit a nemůže se rekonfigurovat jeho vnitřní uspořádání tak, jak tomu je u adaptivního kódování, je i v případě kódování použitém v souborech Mp3 dosaženo jisté přizpůsobivosti vůči vstupním datům. Tato metoda spočívá v tom, že vzorky v granuli podobných vlastností jsou rozděleny do čtyřech regionů, které se zakódují každý zvlášť.

Norma poskytuje pro zakódování vzorků z regionů bigvalues celkem 30 různých Huffmanových tabulek s indexy 0 až 31, tabulky 4 a 14 nejsou uvedeny. Tabulka je vlastně jiná reprezentace Huffmanova stromu. Jedná se o seznam trojic tvořených kódovým slovem a hodnotami x a y. Tyto hodnoty jsou výstupními symboly, dekódováním jednoho slova získáme vždy dvojici hodnot naráz. Ať už je velikost tabulky jakákoliv, výstupní vzorky se vždy pohybují mezi hodnotami 0 - 15. Výstupy tabulky č. 1 jsou všechny možné kombinace dvojic čísel 0 a 1, Huffmanův strom pro tuto tabulku má tedy 2x2 = 4 listové uzly. Oproti tomu strom, odpovídající jedné z největších tabulek, tabulce č. 13, má listových uzlů 16x16 = 256.



kódové slovo	x	y
1	0	0
001	0	1
01	1	0
000	1	1

Obrázek 6: Huffmanova tabulka č.1 a odpovídající strom

Je zřejmé, že rozsah hodnot 0-15 nemůže pro uložení vzorků stačit. Proto mají tabulky definovány další parametr zvaný linbits, který podstatným způsobem navyšuje rozsah použitelných hodnot. Použije se tehdy, pokud se x nebo y dekóduje jako maximální hodnota 15 a parametr linbits je dané nenulové číslo N. Pokud toto nastane, přečte se dodatečných N bitů ze vstupu a ty se interpretují jako hodnota v klasickém doplňkovém kódu bez znaménka. Takto získaný přírůstek se přičte k původní hodnotě. Linbits se uplatňuje jen pro některé větší tabulky a jeho maximální hodnota je 13. Za použití tabulky, kde je linbits na maximu, je tedy maximální dekódovatelná hodnota $15 + 2^{13-1} = 15 + 8191 = 8206$. Nutno poznamenat, že ne všechny tabulky mají unikátní data. Některé tabulky totiž sdílí hodnoty pro dekódování s jinými a liší se od nich pouze rozdílnou hodnotou parametru linbits.

Pro dekódování regionu count1 jsou k dispozici pouze dvě tabulky. Dekódováním jednoho slova z tohoto regionu se získají rovnou čtyři vzorky v, w, x a y. Parametr linbits se na tyto tabulky nevztahuje, výstupní vzorky mohou mít buď hodnotu 0, nebo 1.

4.2.3 Dekódovací algoritmus

Nyní už víme všechno potřebné pro dekódování vzorků a pro rekapitulaci si uvedeme kompletní algoritmus [4] pro regiony bigvalues.

1.Dekódujeme bity pomocí odpovídající Huffmanovy tabulky a nazveme je *x a y*.

2.Pokud *x* = 15 a *linbits* > 0, přečteme počet bitů odpovídající *linbits a* přičteme k *x*. Nyní může mít *x* velikost až 8206.

3.Pokud *x* není 0, přečteme znaménkový bit. Pokud je bit nastaven, *x* = -*x*.

4.Opakujeme kroky 2 a 3 pro *y*.

Algoritmus pro region count1 je obdobný, s tím rozdílem, že se nepoužívá parametr linbits a výstupem jsou 4 vzorky najednou, z nichž pro každý nenulový se čte znaménkový bit.

4.3 Získání scalefactorů a descaling

Před Huffmanovými bity v hlavních datech se objevuje blok, ve kterém jsou uloženy scalefactory. Délka tohoto bloku v bitech, který se označuje jako part 2 length, se pro první granuli vypočítá podle jednoho ze tří vzorců.

1. Pokud je použito dlouhé okno, pak:
   part 2 length = 11 * slen1 + 10 * slen2
2. Pokud je použito krátké okno, pak:
   part 2 length = 18 * slen1 + 18 * slen2
3. Pokud je použita kombinace obou oken (block split flag = 1), pak:
   part 2 length = 17 * slen1 + 18 * slen2

V závislosti na použitém oknu tedy získáme buď 21, 36 nebo 35 scalefactorů. Slen1 a slen2 označují délku scalefactoru v bitech a získají se přečtením hodnot z tabulky na indexu scalefac compress ze struktury sideinfo. Pro druhý kanál se scalefactory přenáší zvlášť a nemají žádnou spojitost se scalefactory z prvního kanálu. Ve druhé granuli se ovšem výpočet délky bloku obsazeného scalefactory liší, poněvadž může sdílet skupiny scalefactorů s první granulí a tudíž se nebudou v druhé granuli přenášet. Výpočet se odlišuje jen v případě použití normálního dlouhého okna, u dalších dvou možností se nic nemění od výpočtu pro první granuli.

$$\begin{aligned} part\,2\,length \;=\; & (\neg scfsi[0] * 6 \;+\; \neg scfsi[1] * 5) \;*\; slen1 \;+ \\ & (\neg scfsi[2] * 5 \;+\; \neg scfsi[3] * 5) \;*\; slen2 \end{aligned} \qquad (2)$$

Scalefactorů pro dlouhé okno je celkem 21 a jsou rozděleny do čtyřech skupin 0 - 5, 6 - 10, 11 - 15 a 16 - 20. Jednobitový příznak scfsi opět z části sideinfo informuje o tom, zda je příslušná skupina scalefactorů přítomna, nebo nikoliv. Nulový bit scfsi značí, že skupina scalefactorů není mezi granulemi sdílena, proto se ve vzorci objevuje negace. Až je známa velikost part 2 length, může se vypočítat, jakou část v hlavních bitech zabírají vzorky v Huffmanově kódu, a to velice jednoduše jako rozdíl hodnot part 2, 3 length z části sideinfo a hodnoty part 2 length.

Dekódování scalefactorů pro dlouhé okno probíhá tak, že se ze vstupu načte 11 hodnot délky slen1 bitů a hned za nimi dalších 10 hodnot délky slen2 bitů, celkem 21 hodnot. Obdobným

způsobem funguje dekódování i pro krátké okno, 18 a 18 hodnot délky slen1 a slen2 bitů dají dohromady 36 scalefactorů.

Situace je ale o něco komplikovanější, pokud je použita kombinace obou oken. Pak se jako první načte 8 scalefactorů pro dlouhé okno délky slen1, následovaných 9 scalefactory pro krátké okno rovněž délky slen1 a nakonec dalšími 18 o délce slen2.

Jakmile jsou k dispozici všechny scalefactory a 576 frekvenčních vzorků z Huffmanova dekodéru, může se přistoupit k fázi zvané descaling. Jejím smyslem je obnovit původní hodnoty vzorků vynásobením určitým měřítkem. V enkodéru jsou totiž vzorky před Huffmanovým zakódováním upraveny tak, aby nabývaly co nejmenší absolutní hodnoty. To vede k použití kratších kódů a jednodušších kódovacích tabulek, což se příznivě projeví na velikosti zakódovaného souboru. Následující vztah se aplikuje na všech 576 vzorků.

$$xr_i = sign(is_i) * |is_i|^{\frac{4}{3}} * 2^{\frac{1}{4}A} * 2^{-B} \quad (3)$$

Výpočet výrazů a a B se liší v závislosti na tom, jaké je použito okno.

$$A_{short} = globalgain[gr] - 210 - 8 * subblock\_gain[window][gr]$$

$$B_{short} = scalefac\_multiplier * scalefac\_s[gr][ch][sfb][window] \quad (4)$$

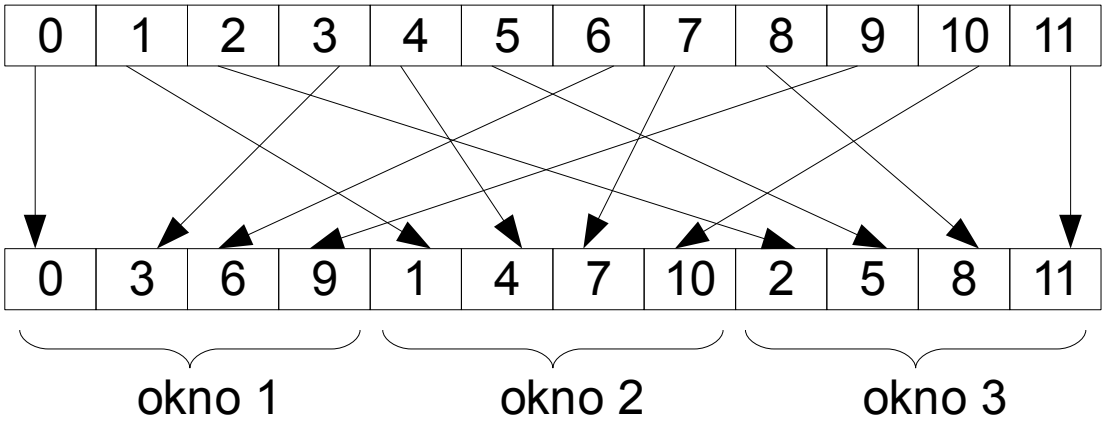
$$A_{long} = globalgain[gr] - 210$$

$$B_{long} = scalefac\_multiplier * scalefac\_l[gr][ch][sfb] + preflag[gr] * pretab[sfb] \quad (5)$$

Hodnota is_i označuje vzorky z Huffmanova dekodéru, xr_i je jeden výsledek fáze descaling. Proměnné globalgain, subblock gain, scalefac multiplier, preflag a pretab se zjistí na základě dekódování struktury sideinfo. Otázkou je, jaké scalefactory se mají do vztahu dosadit pro výpočet výstupních hodnot 1-576. To je dáno tabulkou, která přiřazuje vstupní vzorky k jednotlivým pásmům scalefactorů. Přiřazení vzorků do pásem je ovlivněno vzorkovací frekvencí a použitým oknem.

4.4 Reordering

Reordering je součást dekodéru, která mění pořadí vzorků před vstupem do další fáze. Týká se pouze granulí, kde je použito krátké okno, pro dlouhé okno se nemusí dělat vůbec nic a tato operace se přeskočí. Z důvodu efektivity Huffmanova kódování se v enkodéru vzorky z krátkých oken přeuspořádají podle frekvence, aby byly rozdíly sousedních hodnot co nejmenší. V dekodéru se naopak vzorky v rámci pásma jednoho scalefactoru přeuspořádají podle příslušnosti k oknu. Pokud je použito smíšené okno, prvních 36 vzorků zůstává nezměněno a přeskládávají se až vzorky spadající do čtvrtého pásma scalefactoru pro krátká okna a dále. Obrázek ilustruje přeskládání vzorků v pásmu, které jich obsahuje 12.



Obrázek 7: Přeskládání vzorků

4.5 Dekódování sterea

Po fázi reordering přichází v dekódovacím řetězci jako další na řadu jednotka pro zpracování stereo nahrávek. Nutno poznamenat, že všechny další bloky, navazující na tuto část dekódování (redukce aliasu, IMDCT, atd.), jsou v dekodéru duplikovány a jsou tedy přítomny samostatné bloky se stejnou funkcionalitou, zvlášť pro levý a pravý kanál. Výjimkou pro rozdvojení datové cesty vzorků v této části dekodéru je použití módu mono, který formát MP3 rovněž podporuje. Pak MP3 soubor obsahuje pouze jediný kanál a větev pro zpracování vzorků druhého kanálu zůstává nečinná. Formát MP3 podporuje tyto 4, resp. 5 typů pro zakódování stereo nahrávek:

1. single channel
2. dual channel
3. stereo
4. joint stereo
   4.A middle side stereo
   4.B intensity stereo

Pro připomenutí je způsob kódování sterea určen polem mode v hlavičce každého rámce a je společný pro obě granule. Pokud je vybraným typem joint stereo, pak se přečte z hlavičky i pole mode extension, které stanovuje, který ze dvou druhů joint sterea je aktivní.

První typ single channel odpovídá monofonnímu módu, neexistuje zde druhý kanál. Módy dual channel a obyčejné stereo se ve způsobu dekódování nijak neliší. Oba dva už obsahují dva kanály, rozdíl spočívá spíše v interpretaci těchto kanálů. Zatímco mód stereo má vytvářet prostorový dojem při poslechu, v případě módu dual channel je úmyslem autorů MP3 zaznamenat dvě monofonní nezávislé stopy, které spolu, co se týče zvukové stránky, nijak nesouvisejí. Takový princip se může hodit například pro ozvučení filmu ve dvou jazykových stopách.

Zpracování všech výše jmenovaných módů je velice snadné, jelikož při nich nejsou vzorky nijak modifikovány a jednoduše se přepošlou ze vstupu na patřičný výstup, dle počtu aktivních kanálů. Formát MP3 ovšem počítá se skutečností, že vzorky prvního a druhého kanálu ve stejných časových okamžicích nabývají navzájem blízkých hodnot – je poměrně velká pravděpodobnost, že se hodnoty v obou kanálech nebudou od sebe příliš lišit. Této závislosti využívá poslední mód joint stereo za účelem dalšího navýšení kompresního poměru. Tato komprimace samozřejmě přináší na straně dekodéru zpracování navíc oproti situaci, kdy je použit některý z prvních tří módů.

Mód MS stereo neboli middle-side stereo představuje bezeztrátovou kompresi levého a pravého kanálu. Výstupy pro první (levý) a druhý (pravý) kanál se pak spočítají podle následujících rovnic, kde M_i a S_i jsou vstupní hodnoty pro první a druhý kanál.

$$L_i = \frac{M_i + S_i}{\sqrt{2}} \qquad R_i = \frac{M_i - S_i}{\sqrt{2}} \tag{6}$$

Princip komprese spočívá v tom, že hodnoty Si, spočtené při zakódování jako $(L_i - R_i)$ /sqrt(2), budou za předpokladu, že mezi L_i a R_i není velký rozdíl, nabývat dramaticky menšího rozsahu než hodnoty M_i. To má za následek použití Huffmanovy tabulky s menším rozsahem pro druhý kanál a tím pádem i menším průměrným počtem bitů na zakódování jednoho vzorku.

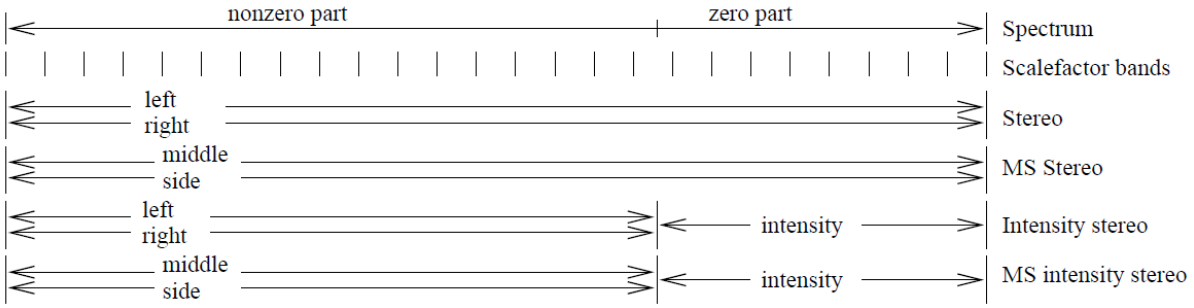
Mód intensity stereo je druhým podtypem módu joint stereo a podobně jako MS stereo redukuje množství informace, přenášené ve druhém kanálu. Poměr přenášené informace v prvním a druhém kanálu je značný, neboť všechny vzorky (jak pro levý, tak i pro pravý kanál) jsou obsažené v prvním kanále a druhý slouží pouze k uložení indexů ispos, které v tomto kanálu nahrazují scalefactory. K samotnému vypočtu hodnot se použije tato rovnice:

$$isratio_{sfb} = \tan\left(ispos_{sfb} * \frac{\pi}{12}\right) \tag{7}$$

$$L_i = L'_i * \frac{isratio_{sfb}}{1 + isratio_{sfb}} \qquad R_i = L'_i * \frac{1}{1 + isratio_{sfb}} \tag{8}$$

Hodnoty ispos ve funkci tangens jsou limitovány pouze na rozmezí 0 až 6, vyhrazená hodnota 7 značí, že dané pásmo není zakódováno technikou intensity stereo a výše uvedené rovnice se na toto pásmo neuplatní. Při důkladnější analýze zjistíme, že hodnoty obou zlomků, kterými se násobí vzorek L'_i z prvního kanálu, nabývají hodnot 0 až 1 a součet těchto zlomků je vždy 1. Zlomky ve výrazu tedy působí jako váhy, které rozdělují „intenzitu“ mezi levý a pravý kanál. Je zřejmé, že tato metoda je na rozdíl od MS sterea ztrátová.

Mód MS stereo má smysl aplikovat pouze na prvních i vzorků, kde i=max(ch[0].rzero, ch[1].rzero) a ch[0/1].rzero je index prvního vzorku kanálu 0, resp. 1, který spadá do oblasti rzero. Jelikož jsou vzorky s indexem větším jak i všechny nulové, rovnají se automaticky zbylé výstupy pro levý a pravý kanál hodnotě nula. Naopak mód intensity stereo se může uplatnit pouze na vzorky s indexem větším jak j, kde j=ch[1].rzero. Dokonce je tedy možné použít zároveň mód middle/side a intensity stereo, jak znázorňuje obrázek.



Obrázek 8: Možnosti nastavení stereo módu[3]

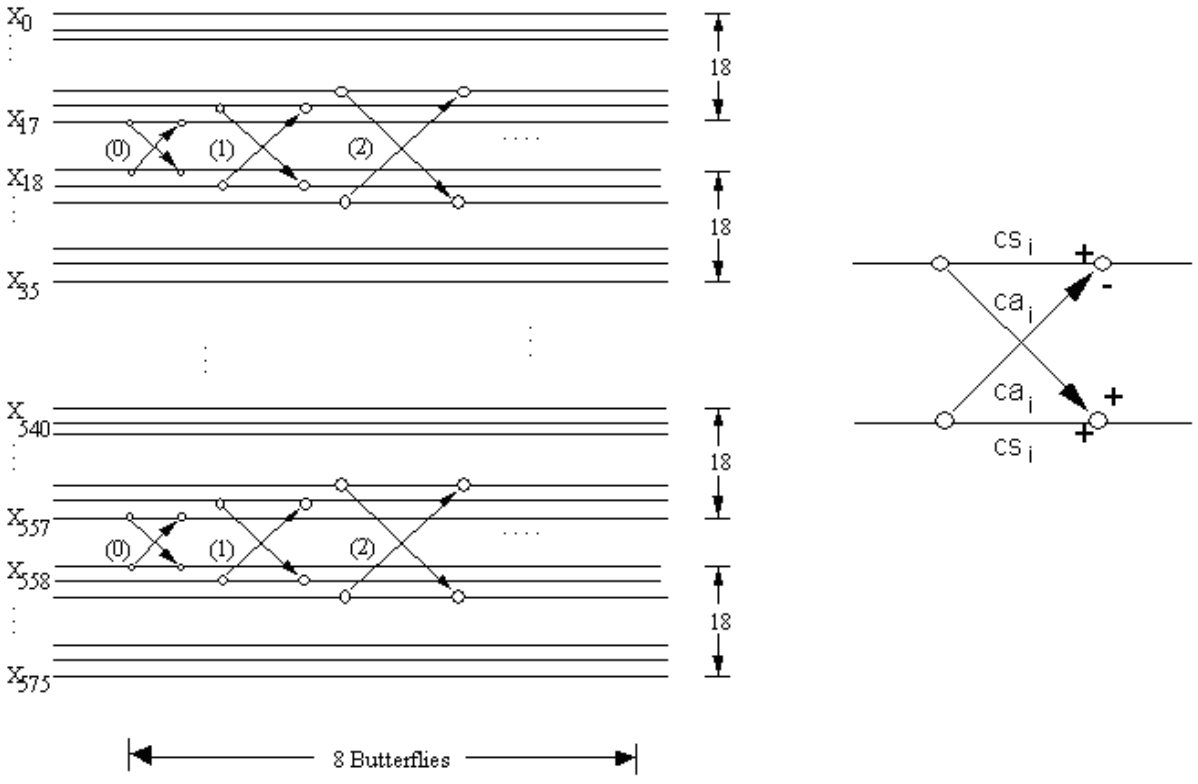
4.6 Redukce Aliasu

V procesu dekódování je nutné před blokem realizující inverzní kosinovou transformaci provést redukci alias efektu. Toho se docílí aplikací 8 motýlkových transformací na každý pár po sobě jdoucích pásem. Zde se označením „pásmo“ ovšem nemyslí skupinu vzorků, sdílejících tentýž scalefactor, nýbrž jedná se o rovnoměrné rozdělení 576 vzorků granule do 32 pásem, každé čítající 18 prvků. Každý „motýlek“ leží na hranici dvou sousedních pásem a je tedy aplikován na spodní polovinu nižšího pásma a na vrchní polovinu pásma vyššího. Proto je prvních a posledních 9 prvků ponecháno beze změny a celkově se provede 31x8 motýlkových výpočtů. Tímto způsobem má být zabráněno vzniku aliasu mezi jednotlivými pásmy.

Všech 31x8 transformací se provede pouze v případě, že jsou použita dlouhá okna. Pro krátká okna není redukce aliasu potřebná a tento krok dekódování se jednoduše přeskočí. Třetí variantou, která může nastat, jsou smíšená okna, kdy je na prvních 36 vzorků aplikováno dlouhé okno a na zbytek vzorků krátké. V tomto případě postačí spočítat pouze jednu řadu osmi motýlků mezi prvním a druhým pásmem.

Jak je vidět z obrázku, jeden motýlek se skládá z operací vynásobení vzorků koeficientem cs_i nebo ca_i a vzájemnému přičtení nebo odečtení těchto součinů. Koeficienty cs_i a ca_i se spočítají podle vzorce 9, kde i je index příslušného motýlku a osm možných hodnot c_i udává norma.

$$ca_i = \frac{c_i}{\sqrt{1 + c_i^2}} \qquad cs_i = \frac{1}{\sqrt{1 + c_i^2}} \tag{9}$$



Obrázek 9: Motýlkové transformace při redukci aliasu[1]

4.7 IMDCT

IMDCT je inverzní transformací k MDCT, neboli modifikované kosinové transformaci. Tato transformace je založena na typu DCT-IV a je specifická tou vlastností, že její výstupy jsou překrývány a sčítány s výstupy transformace, uskutečněné nad předchozí sadou dat, což napomáhá redukovat vzniklé artefakty. Vzorky se v tomto bloku transformují z frekvenční do časové oblasti.

Výpočty prováděné v bloku IMDCT se dají rozdělit do třech kroků [5]. V prvním kroku se spočte samotná IMDCT, podle následujícího vzorce.

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cdot \cos\left(\frac{\pi}{2n}(2i + i + \frac{n}{2})\cdot(2k + 1)\right) \qquad for\ \ i=0..n-1 \qquad (10)$$

Hodnota n se rovná 36, nebo 12, v závislosti na tom, jestli je použito dlouhé, nebo krátké okno. Vstupem transformace je 18(6) vzorků X_k vstupujících z bloku, realizující redukci aliasu. Výstupem IMDCT v tomto tvaru je 36(12) hodnot x_i , čili se počet výstupních hodnot oproti vstupním zdvojnásobí.

Ve druhém kroku se na výstupy x_i z předchozího kroku aplikuje okno – hodnoty jsou vynásobeny váhovací funkcí. Váhovací funkce existují celkem 4, a která se vybere, záleží na daném typu okna.

block type = 0 (normal block)

$$w_i = \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right) \qquad for\ \ i=0..35 \qquad (11)$$

block type = 1 (start block)

$$w_i = \begin{cases} \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right) & for\ \ i=0..17 \\ 1 & for\ \ i=18..23 \\ \sin\left(\frac{\pi}{12}\left(i - 18 + \frac{1}{2}\right)\right) & for\ \ i=24..29 \\ 0 & for\ \ i=30..35 \end{cases} \qquad (12)$$

block type = 2 (short block)

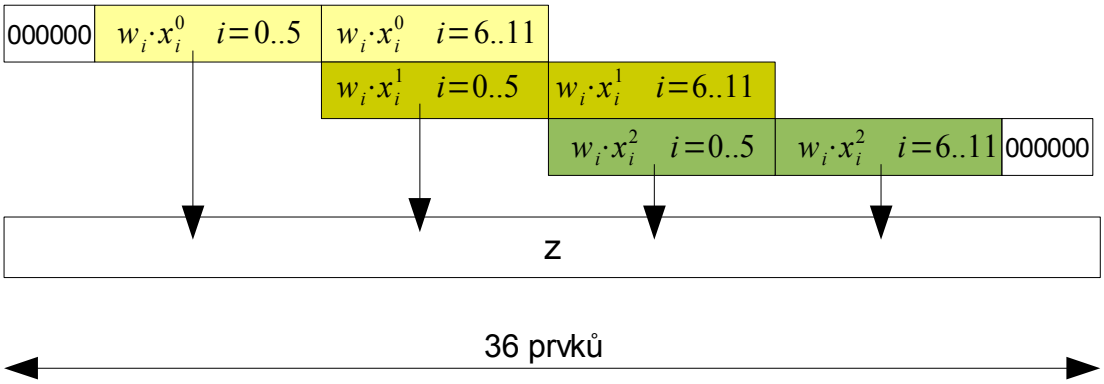
$$w_i = \sin\left(\frac{\pi}{12}\left(i + \frac{1}{2}\right)\right) \qquad for\ \ i=0..11 \qquad (13)$$

block type = 3 (stop block)

$$w_i = \begin{cases} 0 & for\ \ i=0..5 \\ \sin\left(\frac{\pi}{12}\left(i - 6 + \frac{1}{2}\right)\right) & for\ \ i=6..11 \\ 1 & for\ \ i=12..17 \\ \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right) & for\ \ i=18..35 \end{cases} \qquad (14)$$

Pokud se nerovná hodnota block type dvěma, získá se výsledek z_i váhování jednoduše jako součin $z_i = x_i * w_i$, x a w čítají oba 36 prvků. V opačném případě se spočítají tři 6-bodové IMDCT transformace pro 3 krátká okna, jejichž výsledkem je 3x12 hodnot x_i. Tyto hodnoty jsou poté vynásobeny vahami dle vzorce 13, okna jsou přes sebe překryta v polovině jejich délky a sečtena dohromady. Na začátku a na konci se vzniklý vektor doplní šesti nulovými prvky a získá se tak vektor

z o 36 prvcích, který je pro další zpracování ekvivalentní s vektorem z, získaným výše popsaným výpočtem pro block type nerovnající se dvěma.



Obrázek 10: Vytvoření vektoru z pro krátká okna

Třetí dílčí krok, prováděný v bloku IMDCT, je operace overlap-add nad vektory z. Jedná se o podobný výpočet jako při získání vektoru z pro krátká okna. Prvních 18 prvků vektoru z se sečte s 18 posledními prvky *z*-vektoru z předchozího pásma. Druhá, nepoužitá polovina prvků aktuálního pásma se uloží do paměti, aby byla sečtena z prvky z pásma následujícího. Tímto způsobem se spočítá 18 hodnot pro každé z celkem 32 pásem. V následujících fázích dekódování už není zapotřebí udržovat informaci o použitém oknu, protože nadále už není výpočet větven pro typ oken a se všemi vzorky se nakládá stejným způsobem.

Mezi blokem IMDCT a syntézní bankou filtrů se v dekódovacím procesu nachází ještě nachází nepatrný mezikrok, co se výpočetní náročností týče. Jedná se o inverzi frekvencí, kdy se u všech vzorků s lichým indexem, ležících v pásmu s lichým indexem, obrátí znaménko, uvažována je indexace vzorků i pásem od 0.

4.8 Syntézní banka filtrů

Posledním blokem při dekódování souboru MP3 je syntézní banka filtrů, která navazuje na část realizující inverzi vybraných frekvenčních složek. Jejím smyslem je sloučit vzorky z 32 frekvenčních pásem dohromady a rekonstruovat tak původní PCM vzorky. Vstupem v této fázi je 32 vzorků, jedno z každého pásma.

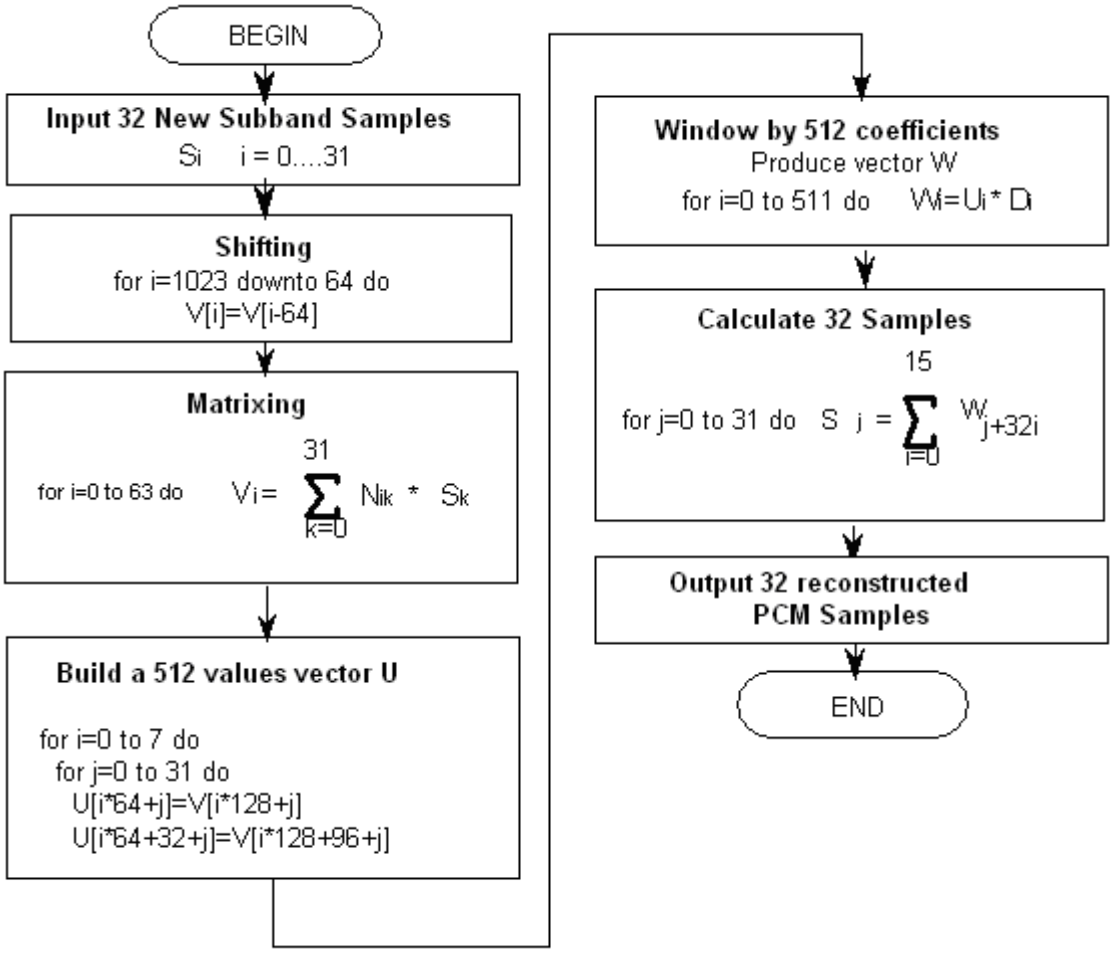
Nad vstupními vzorky se provede další druh IMDCT, pojmenovaný v rámci dekódování MP3 jako matrixing. Hodnoty na vstupu s jsou při ní násobeny členem N_{ik}, který lze dále rozepsat podle následující rovnice:

$$N_{ik} = \cos\left((16 + i)(2k + 1)\frac{\pi}{64}\right) \tag{15}$$

Aplikací operace matrixing získáme 64 prvků, které se uloží na začátek vektoru V. Tento vektor, čítající 1024 položek, funguje jako buffer, obsahující předešlé výsledky matrixingu. Než se na začátek tohoto vektoru uloží nové hodnoty, posune se obsah bufferu doprava a posledních 64 prvků se zahodí.

V následujícím kroku poslouží buffer v ke konstrukci vektoru u o délce 512 hodnot. Z bufferu v se extrahují bloky 64 hodnot s lichými indexy a ty jsou konkatenovány, aby vytvořily vektor U. Sudé bloky se přeskočí a jsou použity při dalším posunu bufferu V. Další vektor, označovaný W, se spočítá jako skalární součin vektoru u s vektorem D. Vektorem D není nic jiného než 512 konstant daných normou, fungujících jako váhovací funkce. V poslední fázi se sečte 16 prvků vektoru W na

odpovídajících indexech, aby vytvořily jeden výsledný PCM vzorek, kterých je spočteno v jedné iteraci celkem 32. PCM vzorky se pošlou na výstup a tím dekódovací proces končí.

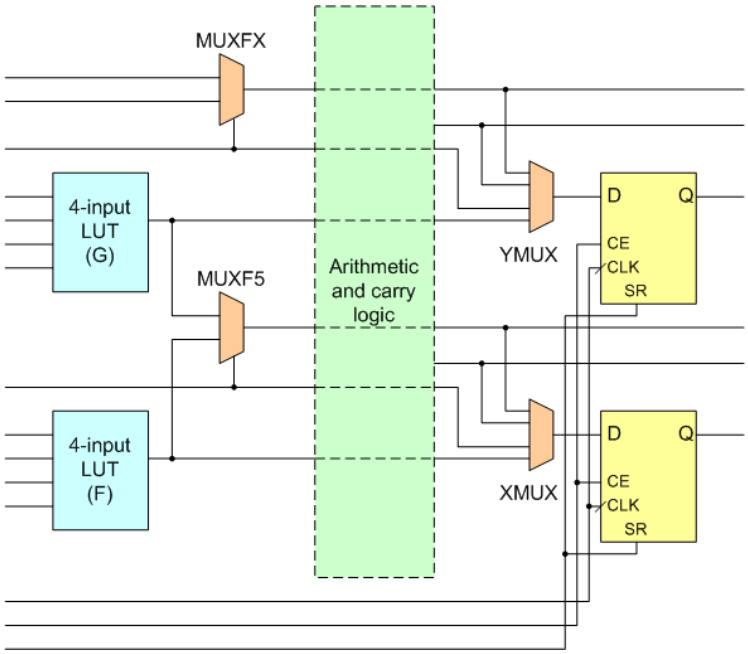


Obrázek 11: Výpočty v syntézní bance filtrů [1]

5 Technologie FPGA

FPGA, neboli Field of Programmable Gate Array, je integrovaný obvod, který je možné naprogramovat uživatelem a změnit tak jeho funkci. Mezi největší výrobce těchto čipů patří společnosti Altera a Xilinx, které produkují rekonfigurovatelné obvody už od 80. let minulého století. Tato technologie nabízí kompromis mezi vysoce výkonným, ale nákladným řešením ASIC (Application Specific Integrated Circuit), a levnějším a relativně pomalejším softwarovým řešením na klasickém procesoru. FPGA se hodí použít spíše pro menší série výrobků, kde je požadavek na krátkou dobu uvedení výrobků na trh, a zároveň tam, kde řešení na CPU nepřináší požadovaný výkon či přijatelnou spotřebu elektrické energie.

Samotný čip se skládá z programovatelných logických bloků, rozvodů hodinového signálu, vestavěných jednotek s pevně danou funkcí (paměti, násobičky, samostatný procesor, DSP bloky) a propojovací sítě. Často bývá využívána tzv. ostrovní architektura, kdy jsou logické bloky uspořádány do matice a propojovací síť zajišťuje pomocí přepínačů požadované zapojení mezi bloky. Programovatelné logické bloky lze dále rozdělit na menší samostatné části zvané slice. Jeden slice obsahuje klopný obvod, multiplexory, funkční generátor a aritmetickou jednotku s vývody pro šíření přenosu do dalšího slice, což se hodí zejména pro implementaci vícebitových sčítaček. Při využití těchto spojů je navíc zpoždění menší, než kdyby se využilo globálních propojovacích vodičů. Funkčním generátorem se rozumí N-vstupá lookup tabulka, která může realizovat libovolnou kombinační funkci N proměnných. Požadované výstupní hodnoty se zapíší do paměti a vstupy pak fungují jako adresové vodiče, vybírající hodnotu z daného řádku tabulky. Lookup tabulky mohou sloužit samozřejmě i jako paměťové elementy, ale i jako posuvné registry.



Obrázek 12: Schéma slice pro Xilinx Virtex-4 [6]

Návrh a implementace FPGA zařízení má několik fází:

- popis chování obvodu
- simulace a testování
- syntéza
- generování konfiguračního řetězce
- naprogramování čipu a provoz

Nejprve se popíše chování obvodu pomocí některého HDL jazyka (Verilog, VHDL). Takovýto model je možné odsimulovat v simulačních nástrojích, jako jsou ModelSim nebo Isim, a verifikovat jeho funkci. Odladěný popis obvodu, jehož správnost se potvrdila při simulacích, se poté podrobí logické syntéze. Za protějšek tohoto kroku ve světě software by se dala označit kompilace programu ze zdrojových kódů. Při syntéze se převádí popis obvodu na funkčně ekvivalentní vyjádření pomocí obecných logických bloků. V následující fázi se obecné logické bloky namapují na konkrétní prvky cílového zařízení a výsledkem je konfigurační řetězec pro daný FPGA čip. Tento soubor se nahraje na čip a provede se jeho naprogramování, čímž je zařízení připraveno k provozu.

5.1 VHDL

VHDL (Very High Speed Integrated Circuit HDL) je programovací jazyk ze skupiny HDL, tedy jazyků pro popis hardware. Jazyk VHDL není spojen s žádnou konkrétní technologií, což jej činí v tomto univerzálním. Umožňuje popisovat obvod na třech možných úrovních – strukturní, behaviorální a dataflow.

Strukturní popis představuje popis obvodu pomocí instancování dílčích komponent a jejich vzájemného propojení. V případě behaviorálního popisu udává funkci obvodu nebo jeho části proces, skládající se z posloupnosti příkazů a konstrukcí, známých z klasických programovacích jazyků, jako je if-else, while nebo switch. Spuštění procesu má za následek nastavení výstupů procesu, jako kdyby se příkazy v těle procesu vyhodnotily sekvenčně. Každý proces obsahuje sensitivity list, neboli seznam citlivých proměnných, který se do jisté míry podobá seznamu parametrů funkce v procedurálních jazycích. Smyslem sensitivity listu je vyvolat spuštění procesu vždy, když se změní hodnota některé proměnné z tohoto seznamu. Třetím způsobem, který VHDL nabízí, je dataflow popis. Při něm se využívají pouze jednoduché přiřazovací příkazy a modelují se datové závislosti mezi objekty. Programování ve VHDL ovšem neznamená výběr pouze jednoho způsobu popisu, ale je možné kombinovat všechny tři druhy dohromady.

5.2 Virtex-6

Virtex-6 je rodina integrovaných obvodů FPGA pocházející od firmy Xilinx, která se objevila kolem roku 2009. Zahrnuje celkem 13 typů, které se od sebe liší jak co do počtu logických buněk, tak i množstvím dostupné paměti a dalších vestavěných jednotek [8]. Pro všechny modely ale platí, že jsou postaveny na 40nm technologii a výrobce pro ně slibuje snížení příkonu oproti předchozí generaci až o 50%. Jako další prostředek pro redukci spotřebované energie je podporován volitelný mód low power, ve kterém je čip napájen 0.9V namísto 1V. Každý slice v programovatelném logickém bloku obsahuje šestivstupou lookup tabulku, což dovoluje implementovat složitější funkce pomocí menšího počtu slice, než by tomu bylo například v případě Spartanu3, jehož slice obsahují pouze čtyřvstupé LUT.

Co se týče dostupné paměti RAM u rodiny Virtex-6, pohybuje se počet pamětí block RAM o velikosti 36kb mezi 156 až 1064 jednotkami. To znamená, že model s nejmenším počtem jednotek disponuje celkem 702kB paměti BRAM, nejvybavenější model může nabídnout 4788kB. Jedna block

RAM paměť o 36kb se dá navíc rozdělit na dvě nezávislé paměti poloviční kapacity. To je výhodné při jemnějším dělení paměti a nedochází tak ke zbytečnému plýtvání prostorem v případě, že se zaplní pouze část paměti block RAM a zbytek zůstává nevyužit.

Mimo jiné je čip pro potřeby zpracování digitálního signálu osazen DSP bloky, zde označovaných jako DSP48E slices (celkem 288 až 2016 slices). U těchto bloků je možné dosáhnout maximální pracovní frekvence 600MHz, stejně jako u programovatelné části obvodu. Jeden DSP slice obsahuje znaménkovou násobičku 25x18 bitů, 48-bitovou sčítačku a 48-bitový akumulátor. Tyto bloky je pak možné použít pro výpočet velice frekventované operace při zpracování signálů, MAC – multiply and accumulate.

Pro usnadnění komunikace s ostatními zařízeními jsou přítomny moduly implementující populární rozhraní Ethernet nebo PCI Express. Počet bloků pro PCI Express je 1-4 a zajišťují provoz až po osmi sériových linkách, což odpovídá teoretické propustnosti až 4GB/s.

5.3 ModelSim

Až je napsán zdrojový kód popisující obvod, je velice vhodné ověřit jeho funkci v některém simulačním nástroji. Takovým nástrojem je například ModelSim, vyvinutý společností Mentor Graphics. Nové verze programu jsou vydávány v několika verzích: ModelSim PE, DE, SE a PE student edition. Verze se od sebe liší jak v počtu funkcí, tak i ve výkonnosti. U typů DE a SE je navíc zaručeno, že bezproblémově poběží na operačních systémech Linux, všechny typy bez výjimky by měly podporovat systém Windows XP/Vista/7. Mezi dříve vydávané verze patřil i ModelSim XE, jenž bylo možné zadarmo stáhnout a používat, nyní již však není dále podporován. Místo toho přichází volně stažitelný ModelSim PE Student edition. Ten je určen pro použití v akademické sféře a pro potřeby studentů k práci na jejich školních projektech, nikoliv však pro komerční využití. Oproti plné verzi PE je omezen celkový počet řádků zdrojového kódu v projektu na 10000 při výkonnosti 30%, projekt obsahující více řádků je sice možné odsimulovat, ale pak už klesá výkonnost na pouhé jedno procento, tedy stokrát pomaleji než v plné verzi [9].

Uživatelské rozhraní ModelSimu je rozděleno do několika oken, jejich velikost a uspořádání si může uživatel sám nastavit podle svých potřeb. Mezi nejdůležitější z nich patří strom projektových souborů, okno textového editoru, příkazová řádka a hlavně okno, do kterého se vykreslují průběhy sledovaných signálů. V tomto okně se lze pohybovat pomocí myši, rychlejší je však využít klávesnici, nabízející zkratky pro přechod na začátek nebo konec simulace, zoom a velice užitečnou funkci, která pro vybraný signál najde nadcházející časový okamžik, ve kterém dojde ke změně hodnoty signálu.

Nejpohodlnějším způsobem, jak spustit simulaci, je připravit si skript fdo, do kterého se umístí všechny potřebné příkazy. Takový skript zpravidla obsahuje tyto příkazy:

- vmap – slouží pro namapování použitých knihoven
- vcom – tímto příkazem se přeloží zdrojové soubory
- vsim – nahraje design a spustí simulátor
- view wave – zobrazí okno, kam se budou vykreslovat monitorované signály
- add wave – vykreslí do výše zmíněného okna hodnoty sledovaných objektů v čase
- run – spustí simulaci probíhající zadaný časový úsek

Při překladu zdrojových souborů pomocí příkazu vcom je nutné dbát na jejich pořadí. Překladový systém vyžaduje, aby byl prvně přeložen zdrojový soubor popisující jednu komponentu a až potom další zdrojový kód, ve kterém je tato komponenta použita. Pokud bychom přirovnali soubory projektu ke stromové hierarchii, pak by platilo, že uzel (soubor se zdrojovým kódem) může být přeložen až tehdy, kdy jsou přeloženi všichni jeho synové (soubory s dílčími komponentami).

I když zdrojové soubory projdou bez problémů syntaktickou analýzou během překladu a z grafu plyne, že kódy jsou korektní i po sémantické stránce, nemusí to nutně znamenat, že daný

obvod půjde vysyntetizovat a nahrát na čip. Simulátor je totiž v tomto ohledu poměrně benevolentní a dovoluje překládat konstrukce, které jinak syntetizovatelné nejsou. Mezi takové patří například smyčky s neznámým počtem iterací nebo funkce pro čtení a zápis do souboru. I tyto konstrukce mají své opodstatnění a velmi se hodí při testování a psaní testbench.

Testbench je zdrojový kód (např. ve VHDL), který je vytvořen speciálně pro účely testování. Svojí syntaxí se neodlišuje nijak zásadně od ostatních zdrojových souborů. Obsahuje entitu, označovanou jako UUT – unit under test, do které jsou vloženy a propojeny všechny potřebné další entity a komponenty. Další podstatnou částí je proces(y), postrádající sensitivity list, jehož smyslem je generovat testovací vektory – vstupní data, jakožto i hodinový signál. V simulátoru se pak ověří, že výstupy odpovídají očekávané odezvě na vstupy.

Nezbytnou částí pro simulaci FPGA komponent jsou instalované knihovny jako unisim, simprim či xilinxcorelib. Nejjednodušší simulátor ModelSim XE má tyto knihovny přeloženy a nainstalovány, pro ostatní verze je ale potřeba tyto knihovny přeložit ručně pomocí příkazu compxlib. To platí zejména pro komponenty vytvořené pomocí programu Coregen. Cesta k adresáři s přeloženými knihovnami se potom přidá do konfiguračního souboru modelsim.ini, pak jsou knihovny viditelné pro všechny projekty, nebo lze mapování provést až dodatečně pomocí příkazu vmap.

5.4 Xilinx ISE a syntéza

Xilinx ISE představuje komplexní sadu programů pro návrh, simulaci, syntézu a programování FPGA zařízení. Toto prostředí ve verzi ISE WebPACK je k dispozici zcela zdarma a je dostupné pro platformy Microsoft Windows i Linux. Jeho výhodou je, že obsahuje všechny potřebné nástroje integrované do jednoho prostředí a není tedy nutné mezi nimi přepínat. Nechybí zde textový editor, simulátor Isim, syntetizátor XST, core generator, šablony jazyka pro různé funkční bloky atd.

Proces implementace má v kontextu prostředí ISE tyto hlavní fáze – synthesis, translate, map, place&route a programming file generation. Vstupem syntézy je uživatelem vytvořený popis obvodu na úrovni registrových přenosů. Další navazující kroky již probíhají automatizovaně.

Během syntézy je HDL popis obvodu převeden na interní reprezentaci, například ve formě logických hradel a klopných obvodů. Takovýto popis se poté optimalizuje, zjistí se, které části obvodu nejsou nikdy využity a je možné je odstranit, nebo které signály nemění svoji hodnotu a mohou být nahrazeny konstantami apod.

Z optimalizovaného popisu vytvoří proces translate netlist, k jehož sestavení použije i netlisty vlastních komponent (vytvořené např. programem coregen), které byly během syntézy považovány za černé skříňky. Proces map má na vstupu netlist kompletního obvodu a jeho úkolem je přiřadit obecným logickým prvkům jejich fyzickou podobu z množiny prvků, kterým dané zařízení disponuje. Jakmile jsou vybrány cílové prvky dané technologie, přichází na řadu jejich rozmístění a propojení v procesu place&route. Volitelným posledním krokem, který má smysl dělat až před testováním obvodu na skutečném zařízení, je generování programovacího souboru.

Výsledkem implementačního procesu je kromě samotného programovacího souboru také soubor statistik. Tyto statistiky uvádějí zejména obsazenou plochu na čipu v počtu slice, maximální dosažitelnou frekvenci obvodu, kombinační cesty, způsobující největší zpoždění či seznam použitých prvků na čipu. Tyto informace dávají programátorovi představu, jak bylo výsledné řešení implementováno.

Do implementačního procesu vstupuje ve skutečnosti kromě HDL popisů ještě soubor obsahující omezující podmínky, v ISE označený jako soubor UCF (user constraints file). Ten specifikuje další požadavky, jako jsou minimální frekvence hodinového signálu, zpoždění na vstupních a výstupních pinech, mapování logických jednotek na fyzické prvky zařízení nebo přiřazení vstupních a výstupních pinů na čipu.

5.5 Coregen

Core Generator, zkráceně coregen, je program s vlastním grafickým uživatelským rozhraním obsažený v balíku Xilinx ISE, který slouží pro návrh systému za použití IP (intellectual property) jader. Těmi se rozumí parametrizované šablony funkčních bloků, které jsou návrháři volně k dispozici. Návrhář pak nastaví parametry dle potřeby, vygeneruje instanci jednotky a zahrne ji do svého návrhu.

Prvním krokem při použití nástroje Coregen je vytvoření nového projektu, během něhož se vybere z nabídky rodina FPGA a přesný typ zařízení, na kterém bude navrhovaný obvod běžet. Připravena je široká paleta IP jader, nechybí zde základní jednotky, jako jsou sčítačky, odčítačky a násobičky, dále trigonometrické funkce, paměťové prvky, FIFO, blokové paměti, makra pro programování aritmetických operací v DSP blocích a v neposlední řadě i moduly pro komunikaci po Ethernetu nebo sběrnici CAN či bloky pro zpracování obrazu.

Po vybrání vyhovující jednotky se spustí průvodce, ve kterém uživatel upřesní dále svoje požadavky. Mezi nastavovanými parametry se často objevuje bitová šířka vstupů a výstupů, přítomnost resetovacího vodiče, inicializační hodnoty pro read-only paměť, počet stupňů pipeline apod. Po nastavení všech parametrů spustí průvodce generování výstupních souborů. Klíčové soubory vznikající v této fázi mají koncovky:

- NGC – tento soubor obsahuje netlist vygenerovaného IP jádra a je nezbytný pro syntézu obvodu
- VHD – zdrojový kód ve VHDL, který je potřeba pouze pro účely simulace
- VHO – je rovněž soubor obsahující VHDL kód, je v něm ukázán příklad inicializace vzniklé komponenty, tedy úsek kódu, který je připraven ke zkopírování do vlastních zdrojových souborů.

Po přidání souborů .ngc a .vhd k projektu a do vlastních zdrojových kódů, jak je ukázáno v souboru .vho, je vygenerovaná jednotka připravena k použití.

6 Návrh

Jak již bylo zmíněno, první částí dekodéru se zabývá čtením souboru MP3. Identifikuje se začátek rámce, přečtou se položky hlavičky a struktury sideinfo, scalefactory a extrahuje se část dat, ve které jsou uloženy Huffmanovy vzorky. Jelikož představuje práce se soubory čistě na úrovni hardware neřešitelný problém, předpokládá se, že bude na čipu FPGA vysyntetizován procesor s vlastním operačním systémem (např. Linux, běžící na soft-core procesoru Microblaze [10]). Program, běžící na tomto procesoru, by pak měl otevřít MP3 soubor a provést výše zmíněné kroky. Zpracování souboru a jeho čtení nejsou tedy předmětem této práce a ani se nepředpokládá jejich implementace v hardware. Procesor komunikuje s hardwarovou částí, Huffmanovým dekodérem počínaje, a krom dat fyzického rámce, obsahujícího Huffmanovy vzorky, posílá i další informace, potřebné v procesu dekódování.

6.1 Huffmanův dekodér

Pro Huffmanův dekodér se nabízí použít jako velice rychlou metodu vyhledávací tabulku. Ta by fungovala na jednoduchém principu, v tabulce by se použilo hledané kódové slovo a hned by se přečetly výstupní hodnoty. Jelikož dopředu nevíme délku daného kódového slova, muselo by se načíst tolik bitů, jaká je délka největšího kódového slova z Huffmanových tabulek. Tím by se ale zanesla ohromná redundance, protože při načtení např. 8 bitů, z nichž první dva tvoří platné kódové slovo, nás zbylých 6 bitů nezajímá (jsou částí následujícího kódového slova). Tabulka pro správnou funkci musí však pokrýt všechny možnosti a proto se v ní objeví 64 řádků, ve kterých se budou opakovat ty stejné výstupní hodnoty. Dalším důvodem, který tuto metodu diskvalifikuje, je fakt, že nejdelší kódové slovo v Huffmanových tabulkách má délku 19 bitů. Vyhledávací tabulka by pak musela obsahovat 2^{19} řádků, což představuje neúnosné množství potřebné paměti. Spíše než klasická vyhledávací tabulka by se zde hodila paměť CAM (content addressable memory), ve které by se hledalo nikoliv podle indexu, ale obsahu kódového slova.

Jiný přístup nabízí průchod Huffmanova binárního stromu. Pro tento algoritmus je nutné vybudovat binární strom, který bude v každém uzlu obsahovat ukazatel na levý a pravý podstrom (mohou být prázdné), příznak, zda se jedná o listový uzel a pokud ano, tak i dekódované hodnoty x a y pro region bigvalues, popř. i v a w pro region count1.

Je jasné, že strom kromě listových uzlů, kde jsou uloženy žádané dekódované hodnoty, obsahuje mnoho uzlů řídících, jejichž smyslem je pouze poskytnout ukazatel na další uzel při průchodu stromu. Způsob vytvoření Huffmanova stromu spojováním symbolů s nejmenší pravděpodobností zajišťuje, že pokud daný uzel není uzlem listovým, pak jsou ukazatele na levý a pravý podstrom platné, tedy ani jeden z podstromů není prázdný. Důsledkem toho je, že poměr počtu listových uzlů k počtu zbylých uzlů je N:N-1, takže pro uložení celého stromu je potřeba uložit cca dvojnásobek počtu listových uzlů. To není z pohledu využití paměťových prostředků oproti předchozímu návrhu špatné.

V referenčním softwarovém dekodéru [5] jsou použita pole 32-bitových hodnot, která vznikla transformací odpovídajících Huffmanových stromů. Jeden prvek pole může vystupovat buď jako ukazatel, anebo jako dekódovaná hodnota. Pokud je prvních 16 bitů nulových, je to příznakem, že se jedná o listový uzel a v nejnižších 2x4 bitech jsou uloženy dekódované vzorky x a y, které mohou nabývat hodnot 0x0 až 0xF. V opačném případě se bity prvku rozdělí do dvou polovin. Prvních 16 bitů se interpretuje jako relativní ukazatel (offset) na levý podstrom. Je to hodnota, která vyjadřuje počet prvků pole, které se musí přeskočit, počítáno od současné pozice, abychom se dostali na prvek, který je kořenem levého podstromu (za předpokladu, že se vstupní hodnotou 0 sestoupíme do levého podstromu, s 1 do pravého). Spodních 16 bitů analogicky představuje ukazatel na pravý podstrom.

Dekódování tedy probíhá tím způsobem, že se začne u kořene stromu (první prvek pole) a podle vstupní hodnoty 0 nebo 1 se pokračuje v levé nebo pravé větvi. Výjimkou je případ, kdy je offset větší než 250. Je to vícenásobný skok, během něhož se nesmí číst bity na vstupu. Následuje VHDL kód, který popisuje možnou podobu pole odpovídající tabulce č.1, a je naznačen postup při dekódování slova „001“.

```
type hufftable_1  is array (0 to 6) of std_logic_vector (15 downto 0);
constant hufftable1: hufftable_t := (

   x"0201", x"0000", x"0201", x"0010", x"0201", x"0001", x"0011"
)
```

Obrázek 13: Ukázka dekódování Huffmanova kódu

Prvky pole mohou být uloženy i na 16 bitech, liché byty jsou totiž nulové a mohou být vypuštěny. Všechny tabulky dohromady mají celkem 2804 16-bitových prvků, to je v součtu 44864 bitů ~ 5608B. Na čipu FPGA je výhodné tak velký blok dat uložit do k tomu určeným pamětem blockRAM. Navíc je možné nakonfigurovat tuto paměť jako ROM (read only memory), což je pro uložení konstant žádaná vlastnost.

6.2 Rekvantizace

V této fázi se pro připomenutí vyhodnocují tyto výrazy:

$$xr_i = sign(is_i) * |is_i|^{\frac{4}{3}} * 2^{\frac{1}{4}A} * 2^{-B} \quad (3)$$

$$A_{short} = globalgain[gr] - 210 - 8 * subblock\_gain[window][gr]$$

$$B_{short} = scalefac\_multiplier * scalefac\_s[gr][ch][sfb][window] \quad (4)$$

$$A_{long} = globalgain[gr] - 210$$

$$B_{long} = scalefac\_multiplier * scalefac\_l[gr][ch][sfb] + preflag[gr] * pretab[sfb] \quad (5)$$

Při bližším seznámení v nich identifikujeme dvě výpočetně náročné části, a těmi jsou umocnění zlomkem 4/3 a dvou mocnin čísla dva, kde se v exponentu vyskytuje proměnná. Operátor umocnění se sice ve VHDL vyskytuje a je funkční v simulacích, ale žádný ze syntézních nástrojů takto náročný operátor neimplementuje. Je tedy nutné poohlédnout se po jiném řešení.

V literatuře [5] je možné najít přístupy, jenž pro výpočet výrazu používají Newtonovu iterační metodu. Rovnice y = x^(4/3) je přepsána do vhodnějšího tvaru $y^3 - x^4 = 0$. Dosazením do vzorce získáme iterační rovnici, jejímž opakovaným vyčíslením se budeme přibližovat skutečnému výsledku.

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} = y_n - \frac{y_n^3 - x^4}{3y_n^2} = \frac{2y_n^3 + x^4}{3y_n^2} \quad (16)$$

Nicméně tento algoritmický přístup je pro implementaci poměrně náročný, vyžadoval by mnoho aritmetických operací, včetně dělení.

Jednodušší řešení nabízí postup, kdy se výraz nepočítá za běhu, ale již známé výsledky pro všechny možné hodnoty na vstupu jsou uloženy v paměti. Pak se hodnota is použije jako index do tabulky s umocněnými hodnotami is a výsledek je k dispozici v jednom taktu. Zbývá určit obor hodnot, jakých může člen is nabývat. Tím jsou všechna celá čísla od nuly až po maximální hodnotu, která může být generována v předcházejícím Huffmanově dekodéru, a tou je 8206. Pokud by byly umocněné hodnoty uloženy na 32 bitech, bylo by nutné pro takovou tabulku vyhradit 8207x32b=32,8kB paměťového prostoru.

Zbývající dva mocněné výrazy z rovnice 3 je možné počítat s výrazem is^(4/3) paralelně, jelikož mezi nimi nejsou datové závislosti. Poněvadž mají obě mocniny společný základ a je mezi nimi operace násobení, lze výraz upravit jako 2^(1/4*A – B). Potom se dá pro výpočet výrazu aplikovat podobný postup jako pro předchozí mocninu, čtení z tabulky. V exponentu se ovšem vyskytují proměnné globalgain(8), subblockgain(3), scalefac_multiplier(1), scalefac(4) a pretab(2), což je celkem 18 bitů pro všechny proměnné celkem. 18-bitový vyhledávací klíč, složený ze všech hodnot dohromady ovšem adresuje neúnosné množství paměti, a proto metodu v této podobě nelze použít.

Řešení nabízí vyhodnocení celého exponentu ještě před vyhledáváním v tabulce. Před exponent se vytkne jedna čtvrtina, takže má potom výraz podobu:

$$2^{\frac{1}{4}(A - 4B)} = 2^{\frac{1}{4}(globalgain - 210 - 8*subblockgain - (4*\text{scalefac\_multiplier}) * (scalefac + pretab))} \quad (17)$$

Vytknutím jedné čtvrtiny před exponent docílíme toho, že člen scalefac_multiplier, který má buďto hodnotu 0.5, nebo 1, je vynásoben 4 a tím pádem je zaručeno, že hodnota exponentu bude spadat do oboru celých čísel (tím je myšlena část exponentu v závorkách, bez vytknuté jedné čtvrtiny). Pak je možné použít hodnotu exponentu jako vyhledávací klíč. Maximální hodnota, kterou může exponent nabývat, je 255 – 210 – 8*0 – 4*0 = 45, minimální je rovna 0 – 210 – 8*7 – 4*1*(15 + 3) = -338. Jak již bylo řečeno, v upraveném exponentu se mohou vyskytovat jedině celá čísla, a celkem se jedná o 45 - (-338) + 1 = 384 unikátních hodnot. Oproti předchozí metodě se díky odstranění redundance zredukoval nemyslitelný paměťový prostor 2^{18} na pouhých 384 položek tabulky.

Vzorce 4 a 5 byly sloučeny do jednoho univerzálního výrazu, který je aplikován jak pro dlouhá, tak i pro krátká okna. V případě, že jsou použita dlouhá okna, stačí nastavit hodnotu členu subblockgain do nuly, při použití krátkých oken se vynuluje člen pretab.

6.2.1 Desetinná čísla

V bloku rekvantizace se jako v prvním objevuje potřeba pracovat i s desetinnými čísly. Předchozí blok, Huffmanův dekodér, je jediná část v celém procesu dekódování MP3, která pracuje čistě s celočíselnými hodnotami. Blok, provádějící rekvantizaci má sice na vstupu také celá čísla, ale pro jeho výstupy to už neplatí. Standardní floating point reprezentace dle IEEE 754 je sice na klasických procesorech běžná, ale to neplatí pro FPGA, které je primárně určeno pro celočíselné výpočty. Realizovat opravdové floating-point operace sice možné je, ale s daleko vyššími nároky na použité zdroje, než by tomu bylo u nativního typu integer.

Výhodnějším typem reprezentace desetinných čísel je fixed point. Ten spočívá v tom, že určitý počet bitů z celkové délky slova je vyhrazen pro desetinnou část a zbylé bity nesou celočíselnou část hodnoty. Řádová čárka má přitom své neměnné, fixní místo. Celá čísla lze do této reprezentace jednoduše převést posunem vlevo o potřebný počet míst. Nevýhodou tohoto principu je

podstatně menší rozsah, než jaký nabízí floating point. Skvělou zprávou ale je fakt, že jednotky, implementující jednoduché aritmetické operace, se nijak neliší oproti klasickým celočíselným jednotkám. Na první pohled se totiž ani nepozná, že se jedná o desetinná čísla, tato skutečnost je známa pouze programátorovi.

6.2.2 Násobení

Kromě potřeby pracovat s desetinnými čísly je blok rekvantizace také první částí v dekódovacím řetězci, kde je vyžadováno násobení. K tomu se nabízí použít vestavěné násobičky na čipu, ty mají šířky operandů 25x18 bitů, včetně znaménka. To ale nemusí vždy stačit, například pro vynásobení dvou 32-bitových hodnot takové násobičky nedokáží v jednom kroku spočítat výsledek.

Řešením je rozdělení operandů a a b délky N na vrchní a spodní část a_h, a_l, b_h a b_l o poloviční délce, tedy N/2. Pak je možné si vystačit i s násobičkou poloviční šířky, na které se provedou celkem 4 násobení operandů délky N/2 - a_hb_h, a_lb_h, a_hb_l a a_lb_l. Ta násobení, kde se vyskytuje člen a_h, jsou se znaménkem, zbylá dvě jsou bezznaménková. Tyto dílčí součiny se následně posunují doleva a rozšiřují na délku 2N bitů. Součin vrchních polovin operandů se posune o N bitů doleva, prostřední členy a_lb_h, a_hb_l se posunou o N/2 bitů a nakonec součin spodních polovin operandů se neposunuje nikam. Ve všech případech se do horních nevyužitých bitů rozšíří znaménko. Sečtením těchto čtyřech mezivýsledků získáme požadovaný součin a a b na 2N bitech. Násobení NxN tak bylo nahrazeno 4 násobeními N/2xN/2, třemi posuvy a třemi součty. Tento postup ovšem nebude fungovat, pokud bude znaménko druhého operandu (násobitele) záporné. Proto je nutné před začátkem výpočtu provést kontrolu znaménka a když je záporné, změní se znaménka obou operandů na opačná.

6.3 Reordering

Blok reordering nemá příliš velké požadavky na výpočetní výkon. Jeho smyslem je přijmout data ze vstupu a uložit je do paměti na příslušný index, který se vypočítá podle výrazu:

$$index = (i * \text{sfband\_size}/3) \ \% \ \text{sfband\_size} + i/3 \qquad (18)$$

Sfband_size je šířka daného pásma scalefactoru velikosti N, přitom je zaručeno, že N je beze zbytku dělitelné třemi. Proměnná i představuje počítadlo uvnitř tohoto pásma a nabývá tak hodnot 0 až N-1. Problematické dělení třemi se dá nahradit vyčtením hodnoty z tabulky, jelikož i může být jen tak velké, jako je maximální šířka sfband_size, a to je 126. Operace modulo je rovněž pro FPGA nepraktická, a proto je vhodnější tuto operaci nahradit čítačem modulo N. Ten se po dosažení maximální hodnoty N-1 překlopí zpátky do nuly.

Na vstupu bloku ale nemáme k dispozici informaci o indexu aktuálního vzorku v rámci granule, natož index uvnitř jednoho pásma. Vstupem je pouze typ okna, vzorkovací frekvence, vzorky a signál, který značí, že vzorky jsou připraveny pro čtení. Je tedy nutné implementovat čítač vzorků, z jehož hodnoty se odvodí aktuální šířka sfband_size, index i v rámci tohoto pásma a další potřebné informace.

6.4 Dekódování sterea

Tato jednotka by měla být schopna dekódovat vzorky z granule, ať už jsou zakódovány pomocí kteréhokoliv stereo módu. Módy single channel, dual channel a stereo nevyžadují žádné zpracování a je možné vzorky okamžitě poslat na výstup. Pro módy intensity a middle/side stereo se už ale aritmetickým operacím nevyhneme.

Ve vzorci 6 pro mód MS stereo se vyskytuje součet a rozdíl dvou vzorků M a S, který se dá jednoduše realizovat pomocí jedné sčítačky a odčítačky. Bylo by zbytečné do návrhu zapojovat složitou děličku kvůli jmenovateli, ve kterém se vyskytuje odmocnina ze dvou. Výhodnější je implementovat násobičku, která vynásobí součet nebo rozdíl hodnot M a S převrácenou hodnotou jmenovatele, tedy 1/sqrt(2). Jelikož je tento násobitel konstantní, dala by se použít i násobička konstantou, která má obecně menší nároky na zdroje, než násobička pro libovolné dva operandy.

$$L_i = \frac{M_i + S_i}{\sqrt{2}} \qquad R_i = \frac{M_i - S_i}{\sqrt{2}} \tag{6}$$

Výpočty v módu intensity stereo vypadají na první pohled složitě, zejména kvůli funkci tangens. Podle normy však parametr může nabývat pouze 7 různých hodnot, a tak se nabízí nahradit výpočet této trigonometrické funkce malou vyhledávací tabulkou. Takto vypočtený parametr isratio je poté použit ve zlomcích, kterými se násobí vzorek z prvního kanálu. Ovšem v těchto zlomcích se kromě samotného parametru isratio vyskytují pouze konstanty hodnoty 1. To nabízí možnost sloučit výpočet tangens a následné vyčíslení zlomků do jedné operace, kterou je vyhledání hodnoty celého zlomku v tabulce na základě vyhledávacího klíče, kterým je parametr ispos. Obsah tabulky je navíc možné díky symetrii redukovat pouze na 7 položek, protože platí $frac_L(ispos) = frac_R(6 - ispos)$, kde $frac_X$ značí vyhodnocený zlomek kanálu X.

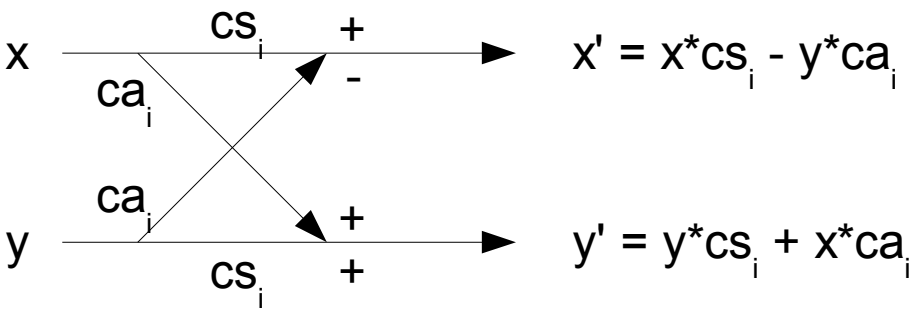
$$isratio_{sfb} = \tan\left(ispos_{sfb} * \frac{\pi}{12}\right) \tag{7}$$

$$L_i = L'_i * \frac{isratio_{sfb}}{1 + isratio_{sfb}} \qquad R_i = L'_i * \frac{1}{1 + isratio_{sfb}} \tag{8}$$

6.5 Redukce aliasu

Jednotka redukce aliasu provádí motýlkové operace naznačené na obrázku. Jelikož předchozí blok, dekódující vzorky obou kanálů, produkuje výsledky sekvenčně, paralelním výpočtem několika motýlků najednou bychom dekódování neurychlili.

Navržený princip tedy přijímá v jednom okamžiku jeden vzorek z celkem 576, obsažených v jedné granuli. Pro výpočet motýlka pochopitelně potřebujeme oba dva vzorky, vrchní x (s nižším indexem) a spodní y. Jako zcela první vzorek z celé granule se pro výpočet motýlka získá vzorek s indexem 10, tedy vrchní vzorek x největšího motýlku. Problém je ale v tom, že v tuto chvíli ještě neznáme spodní hodnotu y, rpo tento případ vzorek s indexem 25, protože vzorky jsou posílány sekvenčně, jejich pořadí je zachováno. Stejná situace nastává i pro další vzorky 11 – 17. Teprve po načtení vzorku s indexem 18, který je spodní hodnotou y nejmenšího motýlku, známe oba dva operandy pro uskutečnění transformace. S načtením vzorků 19 – 25 se provede zbývajících 7 transformací, které dosud čekaly na svůj druhý (spodní) operand y. Stejný postup se aplikuje i na další vzorky, ovšem na patřičných indexech, a spočítá se zbývajících 30x8 transformací. Mezi skupinami vzorků, které jsou použity v sousedních motýlcích, se vždy nachází dvojice vzorků, se kterými není jakkoliv manipulováno a pro výpočet nejsou potřeba – první z nich jsou vzorky s indexy 26 a 27. Stejně tak nepotřebujeme znát pro transformaci hodnoty prvních a posledních deseti vzorků.

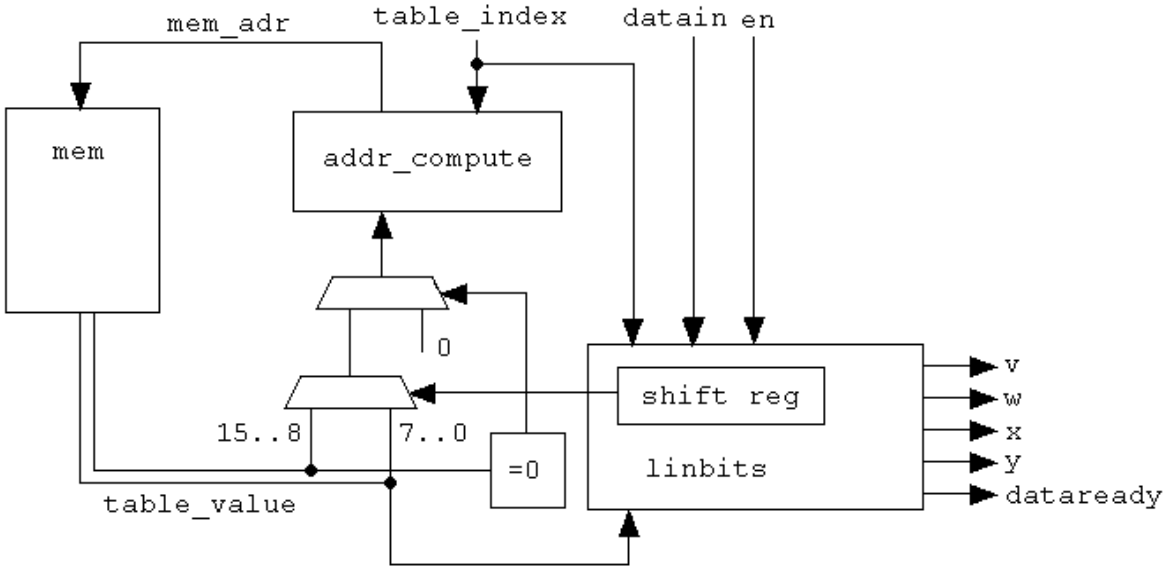


Obrázek 14: Motýlková transformace

7 Implementace

Implementovaný Huffmanův dekodér pracuje dle popsaného algoritmu a prochází Huffmanův strom v podobě tabulek. Aby se zamezilo plýtvání místem, není každá tabulka uložena ve vlastní blokové paměti, ale obsahy všech tabulek jsou spojeny dohromady a tento celek je pak uložen v jedné paměti. Navíc se tím získá i jednotný přístup ke všem tabulkám. Aby takový princip mohl fungovat, musí se pro každou tabulku přiřadit její offset, tedy adresa, na které začínají vlastní data dané tabulky, což se rovná součtu počtu prvků všech předchozích tabulek. Při vyhledávání v tabulkách se pak musí tento offset přičíst k adrese, ze které mají být data čtena.

Zpracování jednoho bitu Huffmanova kódu zabere tři takty. V prvním taktu se spočte adresa dalšího prvku v tabulce sečtením hodnoty předchozí adresy s hodnotou ukazatele na další prvek. Ukazatel musí být vždy před zahájením dekódování nového slova vynulován. Druhý takt zabere vystavení hledaného prvku z tabulky na výstup. Ve třetím taktu se rozhodne podle hodnoty přečtené z tabulky, zda jsme dospěli v Huffmanově stromu k listu, nebo zda se jedná o další nelistový uzel. Pokud je vrchní polovina bitů hodnoty z tabulky rovna 0, pak se jedná o list a ze vstupu se nesmí žádný bit přečíst, ve spodní polovině bitů jsou uloženy dekódované hodnoty x a y (případně v, w, x, y). V opačném případě se jedná o nelistový uzel, kdy vrchní polovina bitů hodnoty z tabulky obsahuje ukazatel na levý podstrom a spodní polovina ukazatel na pravý. Je to přesně ten ukazatel, který se přičítá k adrese v prvním kroku. Teprve tehdy se může ze vstupu přečíst jeden bit, který rozhodne, který z těchto ukazatelů se použije.



Obrázek 15: Schéma Huffmanova dekodéru

Další zpracování probíhá v bloku linbits, kde se vypočte, kolik dodatečných bitů je potřeba přečíst ze vstupu. Tato část obsahuje posuvný registr, kam se ukládají po blocích o velikosti 1B nová data ze vstupu. Tento registr se používá i v předchozí části dekodéru, kdy se rozhoduje, zda se má pokračovat v levém nebo pravém podstromu. Tehdy se čte nejvrchnější bit registru, načež se celý obsah registru posune o jeden bit vlevo a počítadlo platných bitů tohoto registru se zmenší o jedničku. Až se spotřebují všechny bity, zažádá se o nový byte dat nastavením signálu newdataen do hodnoty 1. v bloku linbits je dovoleno posuvnému registru posouvat svůj obsah o vícenásobný počet bitů, podobně jako u komponenty barrel shifter, aby se tak co nejrychleji načetl potřebný počet bitů linbits. Až jsou všechny bity k dispozici, přičte se hodnota linbits k původně dekódovaným členům x a y

a nastaví se správné znaménko. Pak je platnost dat na výstupu signalizována vodičem dataready a může se přistoupit k dekódování dalšího slova.

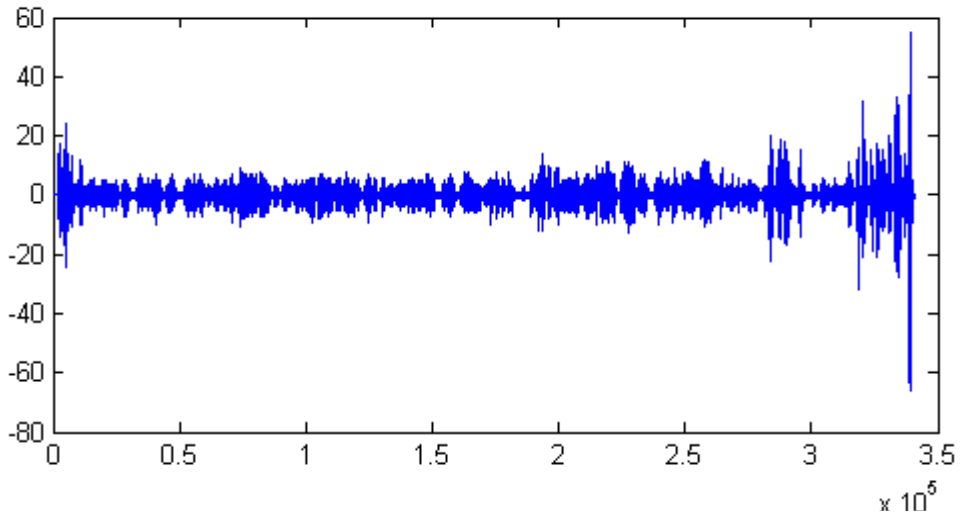
7.1 Rekvantizace

Umocnění Huffmanova vzorku hodnotou 4/3 i mocnina dvojky byly implementovány pomocí vyhledávací tabulky. Nalezené hodnoty z tabulek se poté spolu vynásobí výsledek je k dispozici pro další blok. Latence výpočtu činí dva takty, v prvním taktu se prohledává tabulka a druhý takt zabere násobička. Hodnoty is^(4/3) z první tabulky jsou uloženy ve fixed point reprezentaci (18,14), tedy celkem 32 bitů. 18 bitů pro celočíselnou část bylo zvoleno proto, že maximální hodnota, kterou lze na vstupu dostat, je 8206, přičemž platí, že $2^{17} < 8206^{(4/3)} < 2^{18}$. Stejným způsobem je omezen počet bitů celočíselné části ve prospěch desetinné i u druhého výrazu. Zde byl zvolen poměr celé a desetinné části (12,20), jelikož $2^{11} < 2^{(0.25*45)} < 2^{12}$.

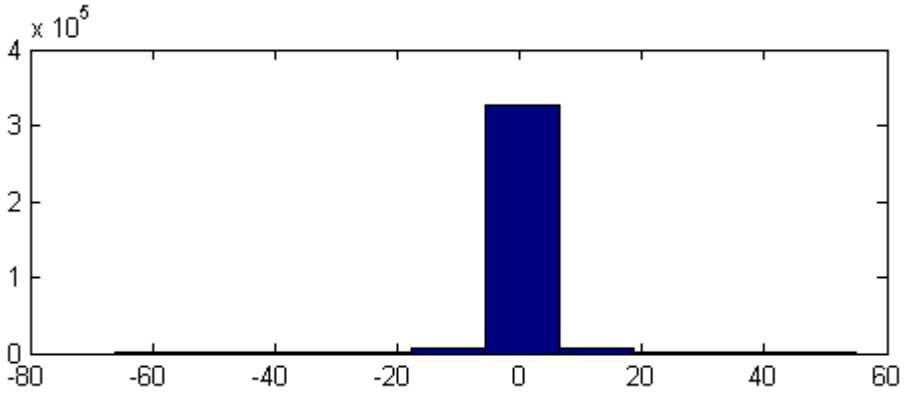
Pro vynásobení obou hodnot je tedy nutné použít násobičku 32x32 bitů. Násobičky zpravidla zobrazují výsledek na dvojnásobném prostoru, než na jakém jsou oba operandy, aby se tak předešlo přetečení, v tomto případě na 64 bitech. Násobička byla vygenerována pomocí nástroje core generator. Z prvního operandu bylo nutné vypustit poslední bit, aby se získalo místo pro znaménko, a zápis se tím změnil na (19,13). Druhý operand nabývá pouze kladných hodnot, takže je možné ho označit jako typ bez znaménka a žádné změny se nemusí provádět. Při násobení desetinných čísel platí, že počet desetinných míst se rovná součtu desetinných míst násobence a násobitele. Z toho plyne, že v 64-bitovém výsledku bude poměr celé a desetinné části (31,33). Z výsledného součinu je vybrána část od 56. po 25. bit, tedy výsledek je ve formě (24,8).

Dále bylo zkoumáno, jaký vliv bude mít omezení počtu záznamů ve druhé tabulce. Plná velikost počítá s 384 záznamy, kde jsou všechny mocniny od 2^(0.25*(-338)) až po 2^(0.25*45). Začátek tabulky tedy tvoří skutečně malá čísla a je vhodné zjistit, zda má vůbec smysl je ukládat, nebo zda je možné zaokrouhlit ty nejmenší hodnoty na nulu.

Pro měření byly vybrány 2 tabulky, jedna ukládá mocniny dvojky na 32 bitech, druhá na 44. Kvůli reprezentaci fixed point se jeví jako nenulových pouze posledních 127, resp. 175 hodnot, zbylé hodnoty jsou zaokrouhleny na nulu. V referenčním software pak byl nahrazen výpočet rekvantizace pomocí funkce pow hledáním v těchto tabulkách a bylo zkoumáno, jaký vliv má tato náhrada na kvalitu dekódovaného souboru



Obrázek 16: Odchylky PCM souboru za použití 127-prvkové tabulky



Obrázek 17: Histogram odchylek

V případě tabulky se 127 prvky se ukázalo, že maximální odchylka se přiblížila v některých místech k hodnotě 60. Vzhledem k tomu, že amplitudy v souboru s PCM vzorky se běžně pohybují okolo 10000, jedná se relativně o malou odchylku. Graf s odchylkami je navíc poněkud zkreslen a lze díky němu nabýt dojmu, že všechny vzorky jsou zatíženy menší či větší chybou. To je způsobeno malým rozlišením, neboť na ose x je vyneseno kolem 350000 hodnot. Ve skutečnosti je chybou zatíženo pouze pár procent vzorků a drtivá většina hodnot má nulovou odchylku, jak ostatně potvrzuje histogram.

Podobné grafy pro druhou tabulku se 175 hodnotami zde uvedeny nejsou, protože na nich toho není moc, co vidět. Odchylky dosáhly v absolutní hodnotě maximálně hodnoty 1, získáváme tedy prakticky stejné výsledky, jako u referenčního řešení. Co se týče posouzení poslechem, jsou vzorky počítané pomocí tabulek a referenční vzorky od sebe nerozeznatelné. Tím se potvrzuje správnost výpočtu rekvantizace pomocí vyhledávání v tabulkách.

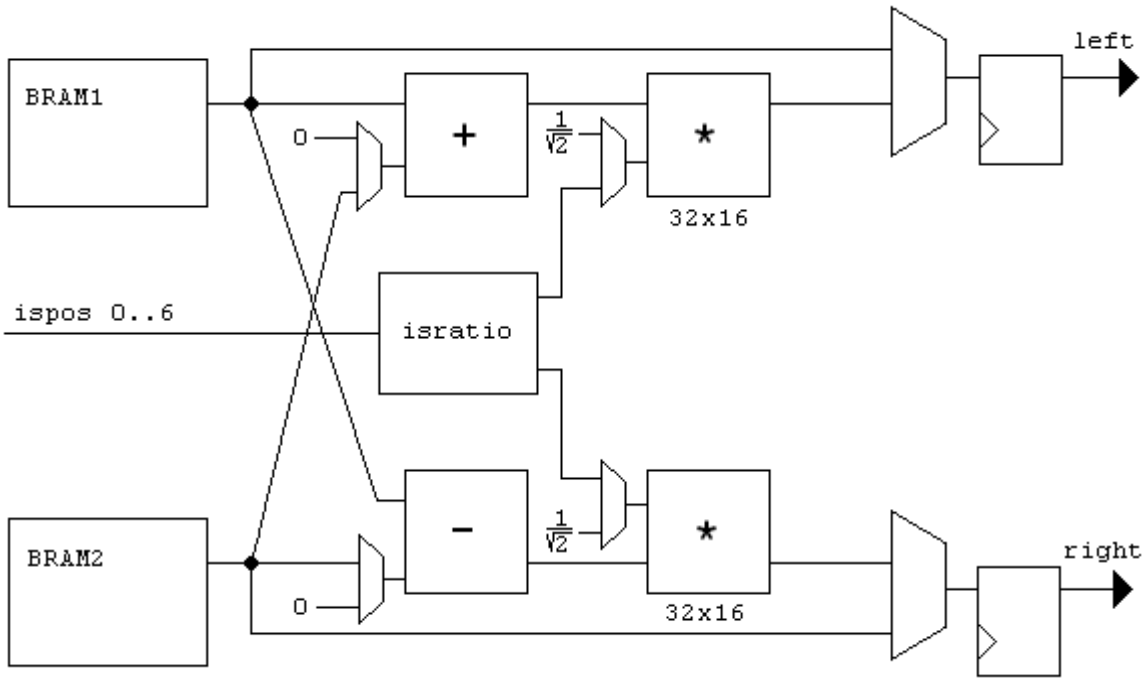
7.2 Dekódování sterea

Blok dekódování byl implementován dle návrhu, přičemž byl kladen důraz na co nejmenší využití zdrojů. Jelikož se v jednom momentu nemohou počítat zároveň módy MS stereo a intensity, je možné prostředky obou módů sdílet, konkrétně se jedná o násobičky.

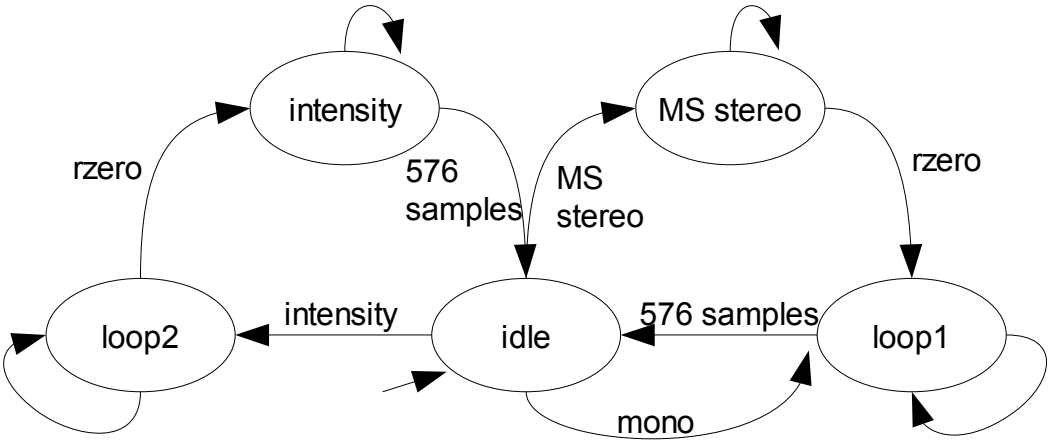
Při módu MS stereo jsou první dva multiplexory nad sebou nastaveny tak, aby na výstup posílaly hodnotu z paměti, tedy M a S. Tím se spočítá v následující násobičce a odčítačce součet, resp. Rozdíl těchto hodnot. Druhý sloupec multiplexorů vybírá v módu MS stereo hodnotu 1/sqrt(2).

V případě módu joint stereo intensity se nastaví první pár multiplexorů pro výběr hodnoty nula, coby neutrální prvek k operacím sčítání a odčítání. Oběma násobičkám je tak nastaven jako jeden z operandů prvek z prvního kanálu. Druhý pár multiplexorů vybírá v tomto módu výsledek z bloku isratio, který přijímá na vstupu parametr ispos, na jehož základě vyhodnocuje hodnotu zlomku ze vzorce 8. Poslední pár multiplexorů přepíná mezi výstupem násobiček a „zkratkou“, po které se přenáší nemodifikované hodnoty, které se načetly ze vstupu. Tento překlenující vodič plní svou funkci, pokud není použitý ani jeden z typů joint stereo. Tato linka musí být zpožděna o jeden takt, aby byla dosažena stejná latence jako u cesty, na které stojí sčítačka a násobička.

Jednotka dekódování sterea je řízena konečným automatem, řízení je zajištěno pomocí generování signálů, připojených na řídící vstupy multiplexorů. V náčrtu diagramu jsou všechny přechody svázány s událostí příchodu nového vzorku na vstupu. První přechod v automatu z počátečního stavu idle je navíc podmíněn i daným stereo módem. Přechod rzero znamená, že byl načten další vzorek a ten patří do regionu rzero. Přechod 576 samples značí načtení všech vzorků z granule.



Obrázek 18: Schéma jednotky dekódování sterea



Obrázek 19: Konečný automat pro Joint stereo

8 Výsledky

Implementované jednotky z předešlé kapitoly byly úspěšně otestovány v simulacích. Kromě nich byla implementována další jednotka, která simuluje čtení ze souboru a posílání hodnot do Huffmanova dekodéru. Byly vytvořeny testovací soubory testbench.vhd, testbench_reorder.vhd a testbench_stereo, které testují zvlášť danou jednotku. U souborů testbench, které simulují chod Huffmanova dekodéru a dekódování sterea, bylo využito možnosti, že jsou výsledky ze simulace zapisovány do souboru. Porovnáním těchto výsledků s výsledky referenčního software se pak snadno zkontrolovalo, zda daná jednotka pracuje dle očekávání.

Bohužel se nepodařilo implementovat všechny jednotky dekodéru, protože jak se ukázalo, jedná se o časově velmi náročný úkol. Jednotky rekvantizace a reordering byly sloučeny do jednoho bloku. Následující tabulka shrnuje použité zdroje a maximální dosažitelnou frekvenci v jednotlivých blocích.

	Slices	Slice registers	Slice LUTs	BlockRAM 36kb, 18kb	DSP bloky	Max. frekvence
Huffmanův dekodér	510/11640	274/93120	1718/46560	1/156	0/288	223MHz
Rekvantizace a reordering	170/11640	350/93120	1263/46560	1/156, 17/312	4/288	135MHz
Stereo decoding	170/11640	223/93120	467/46560	0/156	4/288	135MHz

9 Závěr

V této práci byl nejprve prostudován komplexní formát souboru MP3. Byly rozebrány všechny parametry, potřebné pro úspěšné dekódování souboru, kterých je skutečně mnoho. Dekodér byl rozdělen do několika samostatných bloků, byla popsána jejich funkce a uvedeny výpočty, které se v těchto blocích realizují. Často se výpočty v jednotlivých blocích navíc lišily podle toho, jaký typ okna se použil při fázi kódování, což komplikovalo jednotný návrh.

V krátkosti byla popsána technologie integrovaných obvodů FPGA, jejíž pochopení je důležité pro vhodný návrh obvodu. Dále byly popsány nástroje pro simulaci a syntézu navrženého řešení, jako je ModelSim a Xilinx ISE, a popsány kroky syntézy.

Nejvýznamnější vlastní přínos představuje kapitola pojednávající o návrhu jednotek dekodéru, s ohledem na realizovatelnou implementaci v hardware, popřípadě na konkrétním FPGA čipu Virtex-6. Navržené jednotky byly popsány v jazyce VHDL a jsou plně syntetizovatelné. Jejich korektní funkce byla ověřena porovnáním výstupů ze simulací s výstupy referenčního softwarového dekodéru v jazyce C. Z časových důvodů se bohužel nepodařilo implementovat všechny stupně dekódovacího procesu, takže dekodér nemohl být otestován v praxi.

Jak se ukázalo, implementace přehrávače MP3 pouze v hardware je velice náročný úkol a vyžaduje jisté znalosti v návrhu hardware. Za zvážení jistě stojí, zda se nevyplatí dekodér implementovat softwarově, už z toho důvodu, že drtivá většina operací se zakódovanými vzorky se provádí nad typem floating point, jehož přesnost je na celočíselném FPGA obtížné dosáhnout. Jako další pokračování práce se nabízí implementace zbývajících částí dekodéru, včetně první části, načítající data z MP3 souboru, a otestování dekodéru v praxi.

Literatura

[1] ISO/IEC 11172-3. Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s – Part3: Audio, Švýcarsko: ISO/IEC, 1993. 150 s..

[2] Psychoacoustics. 3. 1. 2012, [online], [cit. 9. 1. 2012]. URL <http://en.wikipedia.org/wiki/Psychoacoustics>.

[3] K. Salomonsen et al., Design and Implementation of an MPEG/Audio Layer III Bitstream Processor, Master's thesis, Aalborg University, Denmark, 1997.

[4] M. Šedivý, Dekodér formátu MP3 pro signálový procesor, diplomová práce, Univerzita Pardubice, Pardubice, 2010.

[5] K. Lagerstom, Design and Implementation of an MPEG-1 Layer III Audio Decoder, master thesis, Chalmers University of technology, 2001.

[6] FPGA Logic Cells Comparison, [online], [cit. 23. 5. 2012]. URL<http://www.1-core.com/library/digital/fpga-logic-cells/>.

[7] Z. Vašíček, Návrh hardware, přednáška předmětu IVH na FIT VUT, 2012, [online], [cit. 23.5. 2012]. URL<https://www.fit.vutbr.cz/study/courses/IVH/private/docs/01-uvod.pdf>.

[8] Virtex-6 Family Overview, [online], [cit. 23. 5. 2012]. URL<http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf>.

[9] ModelSim PE Student Edition description, [online], [cit. 23. 5. 2012]. URL<http://model.com/content/modelsim-pe-student-edition-hdl-simulation>.

[10] Microblaze Linux, [online], [cit. 23. 5. 2012]. URL<http://wiki.xilinx.com/microblaze-linux>.