UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Dekodér formátu MP3 pro signálový procesor

Miroslav Šedivý

Diplomová práce

2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Miroslav ŠEDIVÝ**
Osobní číslo:	**I08342**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Komunikační a řídicí technologie**
Název tématu:	**Dekodér formátu MP3 pro signálový procesor**
Zadávající katedra:	**Katedra elektrotechniky**

Zásady pro vypracování:

Cílem práce je vytvořit program (rutinu) pro signálové procesory řady SHARC (SIMD modely), který bude schopen realizovat dekódování formátu MP3 (MPEG-1/2 layer3) v "CD kvalitě". Cílovou platformou je vývojový kit ADSP-21369 SHARC EZ-KIT. Program by měl být optimalizován pro danou platformu. Předpokládá se, že zdrojová data budou uložena ve SDRAM paměti na kitu, výstup pak bude v reálném čase na zvukový kodek na desce kitu.

Osnova práce:
- popis algoritmu dekódování MP3
- DSP procesory řady SHARC
- vývojový kit ADSP-21369 EZ-KIT lite
- výběr způsobu implementace - volba jazyka, vhodných datových typů.
- vývoj podpůrného programového vybavení - přenos mp3 souborů z PC do SDRAM paměti kitu
- implementace rutiny
- testy

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

Firemní literatura Analog Devices k DSP řady Sharc, dostupné online na http://www.analog.com/en/embedded-processing-dsp/sharc/processors/manuals/resources/index.html
Přednášky z předmětu Signálové procesory (INSPE na FEI UPa)
Standard ISO/IEC 11172-3:1993, dostupné online na http://www.iso.org/iso/catalogue_detail.htm?csnumber=22412

Vedoucí diplomové práce: **Ing. Martin Hájek**
Katedra elektrotechniky

Datum zadání diplomové práce: **15. ledna 2010**

Termín odevzdání diplomové práce: **21. května 2010**

prof. Ing. Simeon Karamazov, Dr.
děkan

L.S.

Ing. Zdeněk Němec, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce, jako školního díla, podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 20. 8. 2010 Miroslav Šedivý

Poděkování

Touto formou bych chtěl poděkovat hlavně vedoucímu mé diplomové práce, panu Ing. Martinovi Hájkovi, za jeho cenný čas a rady, které mi věnoval při řešení problémů spojených s touto diplomovou prací.

Anotace

Tato diplomová práce se zabývá návrhem MP3 dekodéru pro signálový procesor rodiny SHARC firmy Analog Devices. Teoretická část popisuje jednotlivé kroky dekodéru, platformu SHARC a použitý vývojový kit. V praktické části je popsána funkce navržené rutiny a jsou zde popsány možnosti dalších optimalizací a jejich vliv na rychlost algoritmu.

Klíčová slova

MP3 dekodér, DSP, signálový procesor, SHARC

Title

Decoder of MP3 format for signal processor

Annotation

This thesis deals with MP3 decoder for the SHARC family signal processor Analog Devices. The theoretical part describes the partial steps of the decoder, SHARC platform and development kit used. The practical part describes the proposed routine and describes further optimization options and their impact on the speed of the algorithm.

Keywords

MP3 decoder, DSP, signal processor, SHARC

Obsah

Seznam použitých zkratek a značek

A/D	Analogově-digitální převodník
AAC	Advanced Audio Coding
AES/EBU	Audio Engineering Society/European Broadcasting Union
ALU	Arithmetic Logic Unit
CD	Compact Disc
CBR	Constant Bitrate
CRC	Cyclic Redundancy Check
CRC-16	Cyclic Redundancy Check o délce 16 bitů
D/A	Digitálně-analogový převodník
DAG	Data Address Generator
DAI	Digital Audio Interface
DCT	Discrete Cosine Transform
DDR2	Double Data Rate 2
DMA	Direct Memory Access
FCT	Fast Cosine Transform
FFT	Fast Fouirer Transform
FIFO	First in First out
FIR	Finite Impulse Response
FLAC	Free Lossless Audio Codec
GNU/GPL	GNU General Public License
I/O	Input/Output
I2S	Integrated Interchip Sound
ID3	Identify an MP3
IDCT	Inverse Discrete Cosine Transform
IEEE	Institute of Electrical and Electronics Engineers
IIR	Infinite Impulse Response
IMDCT	Inverse Modified Discrete Cosine Transform
ISO	International Organization for Standardization
ISO/IEC	ISO/ International Electrotechnical Commission
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LUT	Lookup table
M/S stereo	Mid-side stereo
MAC	Multiply and Accumulate
MAD	MPEG Audio Decoder
MDCT	Modified Discrete Cosine Transform
MFLOPs	Million Floating Point Operations per Second
MP3	MPEG-1 layer 3
MPEG	Motion Picture Experts Group
MPEG-1	Motion Picture Experts Group version 1

MPEG-2	Motion Picture Experts Group version 2
OOP	Objektově orientované programování
PCM	Pulse-code modulation
PWM	Pulse-width modulation
RCA	Radio Corporation of America
ROM	Read Only Memory
RS-232	Standard pro sériový přenos dat
S/PDIF	Sony/Philips Digital InterFace
SDRAM	Synchronous Dynamic Random Access Memory
SHARC	Super Harvard Architecture
SIMD	Single Instruction Multiple Data
SPI	Serial Peripheral Interface
SRAM	Synchronous Random Access Memory
TDM	Time Division Multiplex
TWI	Two Wire Interface
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VBR	Variable Bitrate
WMA	Windows Media Audio
ZIP	Univerzální bezeztrátová kompresní metoda

Seznam obrázků

Seznam tabulek

1. Úvod

Formát MPEG-1 layer 3, známý též pod zkratkou MP3, je v dnešní době bezkonkurenčně nejznámějším a nejpoužívanějším formátem pro ukládání digitální hudby. Historie formátu MP3 spadá do konce 80. let minulého století, kdy Mezinárodní organizace pro standardy (ISO) dostala za úkol, společně se skupinou MPEG, vyvinout kódovací techniky schopné uchovat komprimované digitální video a zvuk. Část standardu zabývající se zvukem definuje tři módy komprese, které se liší komplexností a také maximálním kompresním poměrem. Tyto módy se nazývají vrstvy (layers). Třetí z nich, layer 3, je právě kompresní technika známá pod názvem MP3.

V době nástupu formátu MP3 bylo primárním nosičem pro ukládání digitální hudby hudební CD. Malé kapacity pevných disků tehdejších počítačů však neumožňovaly ukládání hudby do osobních počítačů ve formátu PCM, který je právě přítomný na hudebním CD. MP3 nabídl řešení tohoto problému. Pro uložení stereo nahrávky bez slyšitelného rozdílu stačilo oproti 1,4 Mb/s z audio CD pouze 128 kb/s. Takto obrovská komprese umožnila rozšíření digitální hudby i do přenosných zařízení. Přenosné MP3 přehrávače používají specializované DSP optimalizované právě na dekódování hudby, dnes už nejen MP3, ale také AAC, nebo WMA. Ačkoliv měly některé procesory osobních počítačů v 90. letech minulého století problém MP3 formát dekódovat v reálném čase, dnes si MP3 pustíte s největší pravděpodobností i na svém mobilním telefonu.

Ačkoliv existuje mnoho dekodérů formátu MP3, žádný z nich není určen pro DSP řady SHARC. Jednou z možností by bylo upravit některý stávající dekodér pro platformu SHARC. Jelikož jsou ale tyto algoritmy vysoce optimalizované, nebylo by díky tomu možné tak podrobně rozebrat celou problematiku dekodéru. Proto byla zvolena druhá možnost – navrhnout dekodér zcela od začátku. Výsledný algoritmus potom například umožní doplnit aplikaci s DSP řady SHARC o kvalitní zvukový výstup s minimálními paměťovými nároky oproti obyčejné PCM.

Kódovací techniky používané v MPEG-1 layer 3 jsou velice sofistikované a dekódovat tento formát vyžaduje mnoho kroků a mnoho matematických výpočtů. Tato práce se snaží přiblížit postup dekódování MP3 formátu v jednotlivých krocích a snaží se poukázat na propracovanost této kompresní techniky. Postup celého procesu dekódování je v první části podrobně rozebrán a je upozorněno i a na možné problémy, které mohou během tohoto procesu nastat. Další část se zabývá popisem použitého DSP, respektive architektury SHARC. Ve třetí části je popsán použitý vývojový kit s procesorem ADSP-21369. Následuje rozbor navržené rutiny, včetně popisu a umístění jednotlivých funkcí. Celou práci zakončují testy rychlosti a přesnosti navrženého algoritmu.

2. Principy ztrátové komprese zvukových signálů

Každá ztrátová komprese zvuku, jako je třeba MP3, je vždy kompromisem mezi množstvím zabraného místa a výslednou kvalitou zvuku. Typicky je při kompresi možné zvolit bitrate a tím i výslednou velikost souboru a kvalitu zvuku. Obecně znamená vyšší bitrate lepší věrnost komprimované zvukové informace. V porovnání napříč kompresními algoritmy to však nemusí být pravda. Modernější, avšak výpočetně náročnější, ztrátový kompresní algoritmus poskytuje lepší zvukovou kvalitu i při nižším bitrate, než starší kompresní algoritmus. Pro příklad může posloužit MP3 a AAC.

Některé zvuky je díky vysoké nahodilosti velmi obtížné kódovat pomocí ztrátové komprese. Dobrým příkladem je potlesk. Pokud je takový zvuk komprimován s nízkým bitrate, mohou se v dekódovaném výsledku objevit zvukové artefakty (zvuky, které nebyly obsaženy ve zdrojovém signálu). Může se jednat například o tzv. pre-echo, kdy se objeví ozvěna zvuku dříve, než zvuk, který ji vyvolává.

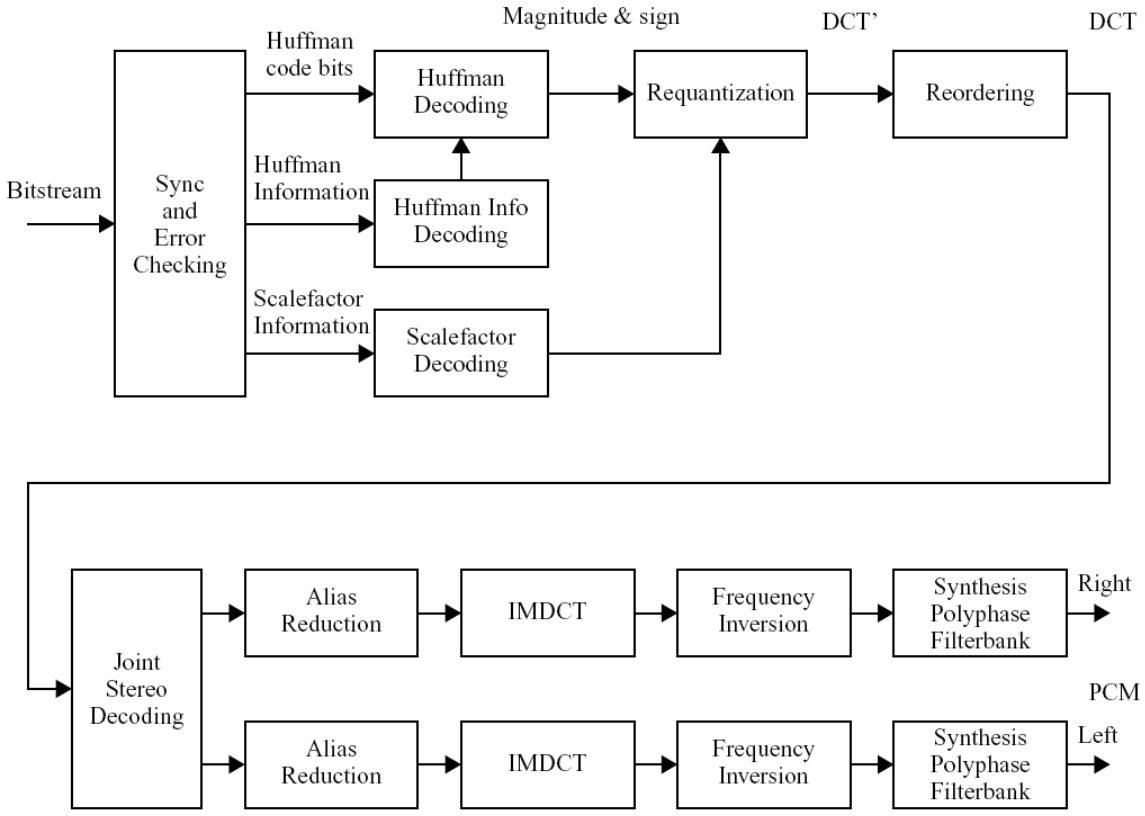
Kvalita výsledného zvuku nezáleží jen na bitrate, ale hlavně na použitém enkodéru. Norma ISO/IEC 1172-3 nepředepisuje konkrétní psychoakustický model, který je použit v enkodéru. Psychoakustický model je matematickým modelem lidského sluchu a určuje jakou informaci je před následnou kompresí možné zanedbat, aniž by posluchač zaznamenal změny oproti původnímu zvuku. Volnost volby modelu lidského sluchu je důvodem odlišné kvality komprese se stejným bitrate, avšak odlišným enkodérem. Kvalita komprese různých zvukových kodeků je hodnocena v poslechových testech, kdy posluchači subjektivně hodnotí kvalitu zvukových kodeků se stejným bitrate [22].

Nejjednodušší typ MP3 používá konstantní bitrate v celém souboru (CBR). Použití konstantního bitrate umožňuje zjednodušit a tím pádem i zrychlit enkódovací proces. Existuje ještě možnost vytvářet MP3, kde se bitrate variabilně mění podle povahy kódovaného zvuku. Takové soubory se nazývají VBR. Určité pasáže, jako je ticho, lze zakódovat pomocí velice malých bitrate a naopak problémové zvuky, kterým je například potlesk, se mohou zakódovat s vysokým bitrate. Velikost souboru je potom podobná, jako v případě CBR, ale věrnost zvuku je mnohem vyšší.

3. Popis algoritmu dekódování MP3

Princip enkódování a dekódování formátu MPEG-1 layer 3 je popsán v normě ISO/IEC 11172-3 z roku 1993 [1]. Norma je třetí částí ze skupiny norem MPEG-1 a zabývá se kompresí zvuku u komprimovaného digitálního videa. V roce 1996 byla norma mírně upravena. Nejednalo se o velké změny, ale pouze o změnu některých termínů, o kterých bude řeč později. Novější norma MPEG-2, popsaná v ISO/IEC 13818-3 , používala audio kompresi zpětně kompatibilní s původním MPEG-1 layer3. Norma MPEG-2 rovněž definuje novější audio kodek AAC, který nahradil MPEG-2 layer 3 a již není zpětně kompatibilní s žádným z formátů MP3.

Formát MP3 je ztrátovou kompresí, takže po dekódování nedostaneme signál identický s originálem. Díky propracovanosti enkodéru, který využívá psychoakustický model lidského ucha, je pro běžného člověka dekódovaný signál k nerozeznání od signálu původního. Norma nenařizuje použití konkrétního modelu lidského vnímání zvuku, takže různé kodeky, kterých existuje desítky, používají různé psychoakustické modely s více, či méně rozdílnými výsledky.



Obrázek 1 - Blokové schéma dekodéru MP3 (zdroj: [8])

Jelikož je MPEG-1 layer 3 nejsložitějším audio kodekem z rodiny MPEG-1, jeho dekódování je velice složité. Díky tomu však nabízí srovnatelnou kvalitu zvuku, i při kom-

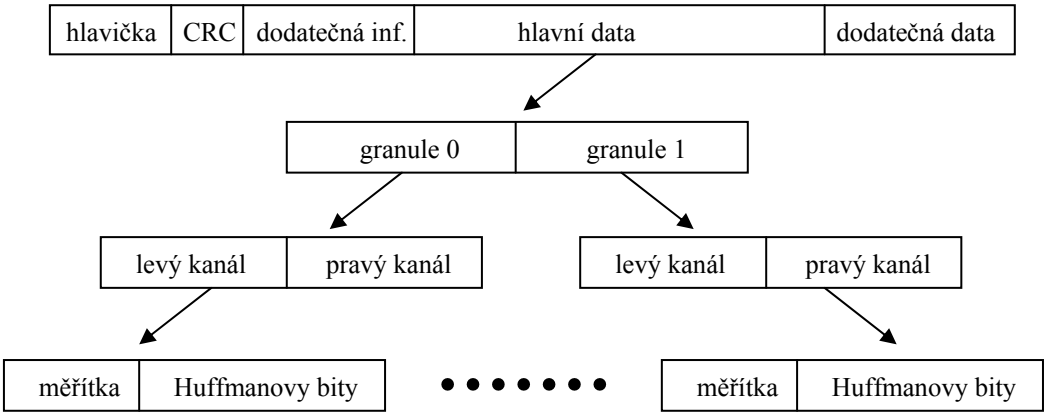
presi až 1:12, v porovnání se zvukem z běžného audio CD ve formátu 16 bit PCM a vzorkovací frekvencí 44100 Hz.

Dekódování se skládá z několika na sebe navazujících kroků, které si nyní popíšeme. Blokové schéma dekodéru je zobrazeno na obrázku 1.

3.1 Synchronizace

Soubor MP3 sestává z nepovinných dodatečných informací, tzv. ID3 tagů, a fyzických rámců s užitečnými daty. ID3 tagy jsou umístěny na začátku souboru a pro dekódování zvukové informace jsou nepodstatné. Obsahují různé informace o názvu zvukového souboru, autorovi, hudebním žánru atd., více informací o ID3 v [2].

Základním krokem při dekódování MP3 souboru je nalezení prvního fyzického rámce. Kvůli informaci ID3, která má různou délku a soubor ji ani vůbec nemusí obsahovat, je poloha prvního fyzického rámce neurčitá a musí se najít. To zajišťuje synchronizační sekvence 11 bitů s hodnotou 1. Tato sekvence je součástí hlavičky, která uvozuje každý fyzický rámec. Hlavička má délku 4 bajty a po jejím dekódování můžeme mimo jiné s jistotou určit, kde se nachází následující fyzický rámec. Díky zvláštnosti formátu MP3, která bude uvedena dále, jsou jednotlivé fyzické rámce mezi sebou závislé, tudíž není možné soubor rozdělit na rámce a dekódovat je samostatně. Obrázek 2 zobrazuje strukturu jednoho rámce. Jednotlivé části budou popsány dále.



Obrázek 2 - Rámec MP3 souboru

3.2 Dekódování hlavičky

Jak už bylo řečeno, hlavička má délku 4 bajty, tzn. 32 bitů. Její zpracování je velice jednoduché a dekódované informace nám dají základní informace o daném souboru. Pro analýzu hlavičky jsou nutné převodní tabulky, které jsou obsaženy v normě [1], v [3], nebo online v [4]. Zde uvedené tabulky nejsou kompletní, ale na rozdíl od originálů obsahují pouze hodnoty pro MPEG-1 layer 3.

3.2.1 Struktura hlavičky

Pro lepší ilustraci jsou jednotlivé bity označené písmeny a bity patřící k sobě mají stejné písmeno. Délka a pozice v následující tabulce jsou uvedeny v bitech.

AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM

Písmeno	*Délka*	*Pozice*	*Popis*
A	11	31-21	**Synchronizační sekvence** (všechny bity musí být nastaveny)
B	2	20,19	Verze MPEG audio - První bit je využit pouze pro rozšíření MPEG 2.5. Pro MPEG-1 a 2 je 1. bit nastaven do 1, takže ve staré normě ISO/IEC 1172-3 byla synchronizační sekvence 12 bitů dlouhá.
C	2	18,17	Identifikace vrstvy (layer)
D	1	16	Ochranný bit, 1 – rámec není chráněn pomocí CRC, 0 – rámec je chráněn a 16 bitů bezprostředně za hlavičkou je vyhrazeno pro CRC
E	4	15-12	Index bitového toku, viz tabulka 2
F	2	11,10	Index vzorkovací frekvence, viz tabulka 3
G	1	9	Padding bit – určuje, zda je rámec doplněn dodatečným bajtem pro dosažení požadovaného bitrate
H	1	8	Privátní bit – pro obecné využití, nemá zvláštní význam
I	2	7,6	Mód kanálů viz tabulka 4
J	2	5,4	Rozšíření módu sterea (pouze pro Joint stereo), viz tabulka 5
K	1	3	Copyright – nastavený bit znamená soubor chráněný autorským zákonem a jeho kopírování je zakázáno
L	1	2	Originál bit – nastavený bit znamená originální nahrávku, nenastavený označuje kopii
M	2	1,0	Emfáze - dává dekodéru informaci pro dodatečnou „reeklvalizaci“ po odstranění šumu. Tato možnost se prakticky nevyužívá.

Tabulka 1 - Význam bitů v hlavičce fyzického rámce MP3 souboru

3.2.2 Tabulky pro dekódování informací z hlavičky

bity	*MPEG-1 layer 3*
0000	volný
0001	32
0010	40
0011	48
0100	56
0101	64
0110	80
0111	96
1000	112
1001	128
1010	160

bity	*MPEG-1 layer 3*
1011	192
1100	224
1101	256
1110	320
1111	neplatný

Tabulka 2 - Převodní tabulka pro bitrate (hodnoty v kb/s)

Volný bitrate znamená, že bitrate je libovolný menší, než nejvyšší povolený. V případě MPEG-1 layer 3 tedy menší než 320 kb/s. Všechny dekodéry nemusejí tuto možnost podporovat. MP3 soubory umožňují i použití VBR, což je proměnný bitrate. Pro dosažení výsledné bitové rychlosti např. 144 kb/s se stále střídají rámce 128 a 160 kb/s. Tuto možnost musí podporovat všechny dekodéry splňující normu ISO/IEC 1172-3.

bity	*MPEG1*	*MPEG2*
00	44100	22050
01	48000	24000
10	32000	16000
11	rezerv.	rezerv.

Tabulka 3 - Převodní tabulka vzorkovacích frekvencí (hodnoty v Hz)

Ve výše uvedené tabulce si můžeme všimnout, že MPEG-1 podporuje pouze 3 možné vzorkovací frekvence, stejně tak v případě MPEG-2, ale vzorkovací frekvence jsou poloviční. Neoficiální rozšíření MPEG-2.5 definuje další vzorkovací frekvence (opět poloviční oproti MPEG-2).

bity	*mód*
00	stereo
01	joint stereo
10	dva nezávislé kanály
11	mono

Tabulka 4 - Převodní tabulka stereo módů

Pro mód joint stereo [5] je ještě dále nutné určit, o jakou verzi joint stereo se jedná. Jak už název napovídá, jedná se o mód, kdy mají oba stereo kanály něco společné.

Bity	*Intensity stereo*	*MS stereo*
00	ne	ne
01	ano	ne
10	ne	ano
11	ano	ano

Tabulka 5 - Rozšíření módu sterea

Výše uvedená tabulka určuje, jakým způsobem je v daném rámci zakódováno stereo. Mód „*Intensity stereo*“ využívá skutečnosti, že člověk nedokáže u některých frekvencí (nízkých a velmi vysokých) určit směr odkud se šíří, takže enkodér zakóduje pouze mono

signál a malou okrajovou informaci s rozdílovým kanálem. Tato možnost nikdy nevede k dekódování původního sterea a může vést i k vytvoření artefaktů.

Oproti tomu „*M/S stereo*“ využívá stejného principu jako kódování stereo informace ve vysílání FM rádia. V jednom kanálu se přenáší součtový a v druhém kanálu rozdílový signál. Tento mód nikdy nezpůsobuje artefakty a je používám i u bezeztrátových audio kodeků (např. FLAC).

3.2.3 Výpočet délky rámce

Po úspěšném dekódování hlavičky máme možnost vypočítat délku aktuálního rámce a tím pádem známe i polohu rámce následujícího. Pro výpočet délky platí vztah 3.1 (operace dělení je celočíselná, dosazujeme v základních jednotkách):

$$délka\ rámce = \frac{144 * bitrate}{vzorkovací\ frekvence} + padding\ bit \tag{3.1}$$

Pro příklad, MP3 soubor se vzorkovací frekvencí 44100 Hz a bitovým tokem 128 kb/s má délku rámce rovnou 417, respektive 418 bajtů s nastaveným padding bitem.

3.3 Detekce chyb

Pokud není nastaven ochranný bit, znamená to, že bezprostředně za hlavičkou se nachází 16 bitů ochranného kódu. Jelikož je formát MP3 výhradně bitově orientovaný, jakákoliv změna, byť jednoho bitu, může kompletně změnit výsledek dekódování.

V MPEG-1 layer 3 je použita detekční metoda CRC-16. Polynom k výpočtu této metody je uveden ve vztahu 3.2.

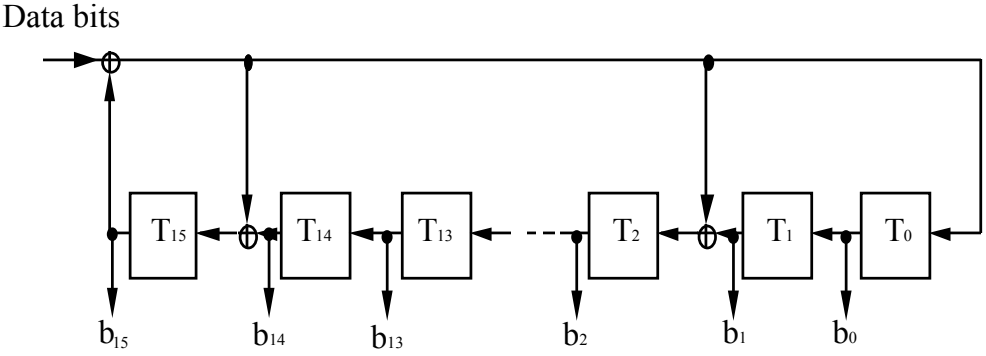
$$x^{16} + x^{15} + x^2 + 1 \tag{3.2}$$

Polynom zvolené CRC ochrany je tedy 0x8005. Je třeba podotknout, že nejvyšší mocnina polynomu (x^{16}) není v hodnotě polynomu obsažena, ale vyplývá z ní chování samotného algoritmu výpočtu. Více o generování CRC v [6].

Chráněny nejsou však všechny bity, ale jen ty nejdůležitější. V případě MPEG-1 layer 3 jsou to bity 16-31 dodatečné informace a bity 0-135 audio dat v případě jednokanálového módu, nebo bity 0-255 audio dat v případě ostatních módů [7].

Princip této metody je zobrazen na obrázku 3. Počáteční stav posuvného registru je '1111 1111 1111 1111'. Následně jsou všechny bity, které je nutné chránit, přiloženy na vstup obvodu z obrázku 3. S každým vstupním bitem se posuvný registr posune o jeden bit. Po zpracování posledního bitu tvoří výstupní bity b_{15} až b_0 datové slovo, které je možné porovnat s ochrannými bity v MP3 datovém proudu. Pokud nejsou obě datová slova stejná,

došlo k chybě při přenosu. Aby se předešlo nepříjemným zvukům, které by mohly vzniknout při dekódování poškozené informace, dekodér může ztlumit výstupní data aktuálního rámce, nebo zopakuje rámec předešlý.



Obrázek 3 - Generování ochranného kódu (zdroj: [7])

3.4 Dekódování dodatečné informace

Dodatečná informace (anglicky *side information*) je umístěna bezprostředně za hlavičkou, v případě rámce chráněného pomocí CRC je to až za dvěma bajty vyhrazenými na ochranu. Dodatečná informace obsahuje všechna důležitá data pro nalezení a dekódování audio dat. Její délka je 17 bajtů pro mód s jedním kanálem a 32 bajtů pro módy s dvěma kanály. Následující obrázek znázorňuje strukturu dodatečné informace pro oba případy (ch určuje index kanálu).

Jednokanálový mód – 17 B

main_data_begin 9 bitů	private_bit 5 bitů	scfsi[ch][scfsi_pásmo] 4 bity	gr0 dodatečná inf. 59 bitů	gr1 dodatečná inf. 59 bitů

Dvoukanálový mód – 32 B

main_data_begin 9 bitů	private_bit 3 bity	scfsi[ch][scfsi_pásmo] 2 x 4 bity	gr0 dodatečná inf. 2x 59 = 118 bitů	gr1 dodatečná inf. 2x 59 = 118 bitů

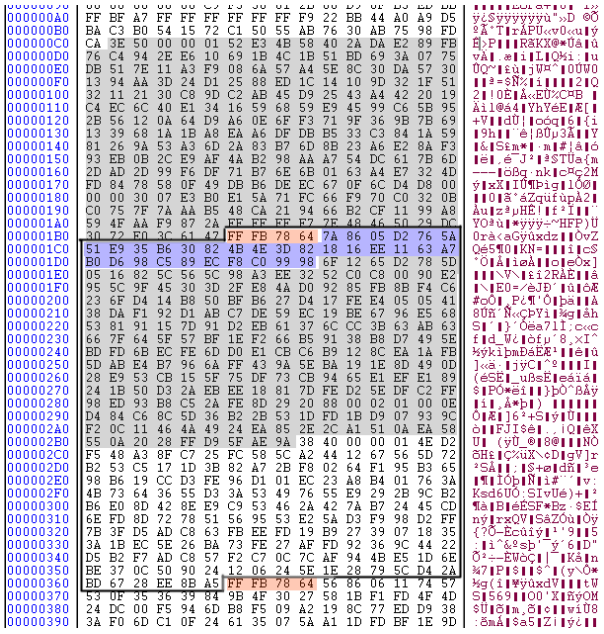
Obrázek 4 - Struktura dodatečné informace

3.4.1 Části dodatečné informace

Pole obsažená v dodatečné informaci budou nazývána originálními anglickými názvy podle normy [1], v závorce za názvem pole je uvedena délka pro mono a následně délka pro mód s dvěma kanály. Toto číslo může být upřesněno v textu.

main_data_begin (9/9)

Určuje negativní ofset začátku hlavních dat od adresy hlavičky v bajtech. Zde narážíme na další pojem - logický rámec. Logický rámec je zcela odlišný od rámce fyzického a jeho začátek je určen právě hodnotou tohoto pole. Pro lepší ilustraci poslouží obrázek.



Obrázek 5 - Struktura MP3 souboru

Na obrázku 5 je znázorněn rozdíl mezi jednotlivými typy rámců. Fyzický rámec je ohraničen tlustou černou čárou. Na jeho začátku jsou 4 bajty hlavičky a 32 bajtů dodatečné informace, tzn., že se jedná o MP3 s dvěma kanály. Logický rámec je vyznačen šedým podkladem a jeho začátek je určen hodnotou pole *main_data_begin*. Konec logického rámce je jeden bajt před začátkem následujícího logického rámce.

Důvodem tohoto dělení je lepší využití místa. Pokud enkodér nepotřebuje pro zakódování informace všechny bity, které nabízí fyzický rámec, použije jich jen část a zbytek se může využít pro následující rámec, kde mohou probíhat dynamické změny zvuku, a samotná délka fyzického rámce by nestačila. Logický rámec totiž nikdy nesmí přesahovat za konec fyzického rámce.

Maximální hodnota *main_data_begin* je 511, takže může nastat i případ, kdy je logický rámec umístěn i o několik fyzických rámců zpět. Nejkratší možná délka fyzického rámce je pouhých 96 bajtů a nastane pro 32 kb/s a vzorkovací frekvenci 48000 Hz.

Private_bits (5/3)

Toto pole je pro obecné použití a při dekódování nemá zvláštní význam.

Scsfi (4/8)

Scale factor selector information – 4 jednotlivé bity pro každý kanál určují, jestli se měřítka pro danou skupinu subpásem (viz tabulka 6) přenáší v každé granuli (bit nulový), nebo jestli jsou společné pro obě granule (bit nastaven). Společná měřítka umožní využít ušetřené bity pro zakódování frekvenčního spektra.

skupina	*subpásma*
0	0,1,2,3,4,5
1	6,7,8,9,10
2	11,12,13,14,15
3	16,17,18,19,20

Tabulka 6 - Skupiny subpásem pro scsfi

Při použití krátkých oken (*block_type* = 2) v jakékoliv granuli se vždy přenášejí měřítka samostatně.

Tři výše uvedená pole jsou společná pro obě granule. V tabulce 7 je struktura polí v jedné granuli. Výskyt polí s šedým pozadím je podmíněný.

part2_3_length	big_values	global_gain	scalefac_compress
windows_switching_flag	block_type	mixed_block_flag	table_select
subblock_gain	region0_count	region1_count	preflag
scalefac_scale	count1table_select		

Tabulka 7 - Struktura okrajové informace jednotlivé granule

Part2_3_length (12/24)

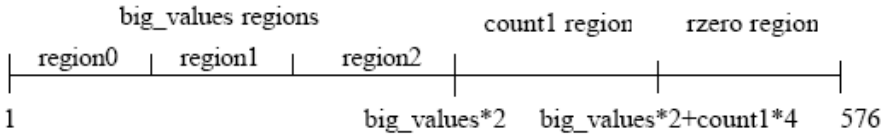
Hodnota vyjadřuje délku 2. a 3. části hlavních dat v bitech (měřítka a bity zakódované Huffmanovým kódem). Po dekódování počtu bitů určeného tímto polem následuje druhý kanál, respektive další granule v případě módu s jedním kanálem. Díky této hodnotě rovněž dokážeme určit začátek dodatečných dat, pokud se v souboru vyskytují.

Big_values (9/18)

Hodnota vyjadřuje počet dvojic frekvenčních čar z regionu *big_values*. Po dekódování každé ze dvou granulí dostaneme frekvenční spektrum složené z 576 čar. Tyto čáry jsou ale kódovány podle různých Huffmanových tabulek, aby byla zajištěna nejlepší komprese. Frekvence od 0 do Nyquistovy frekvence jsou proto rozděleny do pěti oblastí.

Enkodér provádí dělení do skupin podle maxima kvantovaných hodnot. Nízké frekvence mívají většinou vysokou amplitudu a vysoké frekvence mají amplitudu většinou

velice malou, nejvyšší frekvence nemusí být v určitém pásmu vůbec zastoupeny. Rozdělení frekvenčního pásma zobrazuje obrázek 6.



Obrázek 6 - Rozdělení frekvenčního spektra na regiony (zdroj: [8])

Big_values region je dále rozdělen na další 3 části. V každé z nich může být použita jiná Huffmanova tabulka a každá část může obsahovat jiný počet spektrálních čar, případně vůbec žádné. Dekódováním části *big_values* dostaneme dvojnásobný počet frekvenčních čar, než je hodnota *big_values*, podobně v *count1 regionu* dostaneme čtyřnásobek čar. Vysvětlení je v části 3.7 o dekódování Huffmanových bitů.

Délka *count1 regionu* je rozdíl mezi 576 a počtem spektrálních čar dekódovaných z *big_values*. Jestliže už není s ohledem na *part2_3_length* dostatek bitů na dekódování dalších hodnot z *count1 regionu*, zbytek čar do 576 bude mít nulovou hodnotu a tím vznikne *rzero region*.

Global_gain (8/16)

Tato hodnota je použita v rekvantizačním bloku dekodéru pro obnovení původních amplitud spektrálních čar. Jak název napovídá, je pro celé spektrum stejná. Více o *global_gain* v části 3.8 o rekvantizaci.

scalefac_compress (4/8)

Určuje počet bitů vyhrazených pro zakódování jednotlivých měřítek. Granule může být rozdělena do 12, nebo 21 měřítkových subpásem (neplést si s 32 subpásmy po 18 vzorcích). Pokud jsou použita dlouhá okna (*block_type* = {0,1,3}), granule je rozdělena do 21 měřítkových pásem. Při použití krátkých oken (*block_type* = 2) je granule rozdělena do 12 pásem. Měřítka jsou dále rozdělena do dvou skupin, 0-10, 11-20 pro dlouhá okna a 0-6, 7-11 pro krátká okna.

Hodnota určuje index v tabulce. Slen1 a slen2 udává počet bitů použitých pro zakódování velikosti první a druhé skupiny měřítek.

Scaleafc_compress	*slen1*	*slen2*
0	0	0
1	0	1
2	0	2
3	0	3
4	3	0

Scaleafc_compress	*slen1*	*slen2*
5	1	1
6	1	2
7	1	3
8	2	1
9	2	2
10	2	3
11	3	1
12	3	2
13	3	3
14	4	2
15	4	3

Tabulka 8 - Převodní tabulka pro scalefac_compress

Windows_switching_flag (1/2)

Při nastaveném flagu granule obsahuje pole označená v tabulce 7 šedým podkladem. Flag označuje, že je v granuli použité jiné, než dlouhé okno. Kromě toho je při nastaveném flagu délka regionu2 v *big_values* vždy nulová a tím se několik bitů v dodatečné informaci ušetří. Viz *table_select*.

Block_type (2/4)

Toto pole je využito, pouze pokud je nastaven windows_switch_flag a označuje typ okna použitého pro konkrétní granuli. Hodnota 0 je zakázaná, protože značí dlouhé okno. Typy oken podle *block_type* jsou v tabulce 9.

block_type	*typ okna*
0	zakázaná hodnota
1	start
2	3 krátká okna
3	stop

Tabulka 9 - Typy oken

Krátké okno znamená, že 576 čar je rozděleno do 3 oken po 192 vzorcích. Rozložení vzorků v krátkých oknech je z různých důvodů odlišné od rozložení dlouhých oken. Důvod této odlišnosti je upřesněn v 3.9.

mixed_blockflag (1b/2b)

Používá se, jen pokud je nastaven *windows_switching_flag*.

Mixed_blockflag označuje, že jsou použity rozdílné typy oken pro nízké a vysoké frekvence. Pokud je flag nastaven, dvě dolní subpásma jsou transformována pomocí dlouhého okna a zbylých 30 subpásem je transformováno pomocí okna určeného v *block_type*.

table_select (10/20, nebo 15/30)

Standard definuje 30 tabulek, respektive 16 tabulek s různými parametry. Více v části 3.7. Hodnota tohoto pole udává jakou Huffmanovu tabulku použít pro dekódování konkrétního regionu. Délka je 5 bitů (32 různých hodnot pro každý region, granuli a kanál). *Table_select* určuje pouze tabulky pro dekódování části *big_values*. Vybraná tabulka závisí na lokálních statistikách signálu.

Jak bylo zmíněno výše, když je nastaven *windows_switching_flag*, region2 má nulovou délku, takže pouze 2 regiony obsahují zakódované hodnoty. To znamená, že v mono módu stačí na výběr tabulek pouze 2*5 = 10 b. Pokud jsou použity všechny regiony (*windows_switching_flag* = 0) je pro tabulky zapotřebí 15 b na každý kanál a granuli.

subblock_gain (9/18)

Používá se, jen pokud je nastaven *windows_switching_flag* a když je *block_type* = 2, ačkoliv je přenášen nezávisle na *block_type*. Tato 3bitová proměnná označuje ofset zesílení jednotlivých krátkých oken od *global_gain*.

region0_count (4/8), region1_count (3/6)

Region0_count a *region1_count* obsahuje číslo o jedno menší, než je počet měřítkových subpásem obsažených v regionu 0, respektive v regionu1. Hranice regionu jsou nastaveny tak, aby rozdělily frekvenční spektrum do měřítkových subpásem. Pokud jsou použita krátká okna, délka každého okna je vypočítána. Např. když *region0_count* = 8, tak je v něm 9/3 = 3 měřítkových subpásem. Pokud by bylo použito dlouhé okno, obsahoval by region 0 v našem případě 9 měřítkových subpásem. Délka měřítkových subpásem je dána tabulkou uvedenou v normě a je závislá typu bloku a na zvolené vzorkovací frekvenci.

Preflag (1/2)

Zkratka pro dodatečné zesílení vysokých frekvencí kvantovaných hodnot. Pokud je flag nastaven, hodnoty definované tabulkou (Tabulka 10) jsou přičteny k příslušným měřítkům pro jednotlivá subpásma. Pokud je *block_type* = 2 (tzn. krátká okna), *preflag* se nepoužívá.

Subpásmo měřítek	*hodnota*
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	1
13	1
14	1
15	1
16	2
17	2
18	3
19	3
20	3
21	2

Tabulka 10 - Hodnoty pro preemfázi

scalfac_scale (1/2)

Měřítka jsou logaritmicky kvantována s krokem závislým na *scalefac_scale*. Tabulka 11 určuje násobitele, který je použit v rekvantizační rovnici.

scalefac_scale	*scalefac_multiplier*
0	0,5
1	1

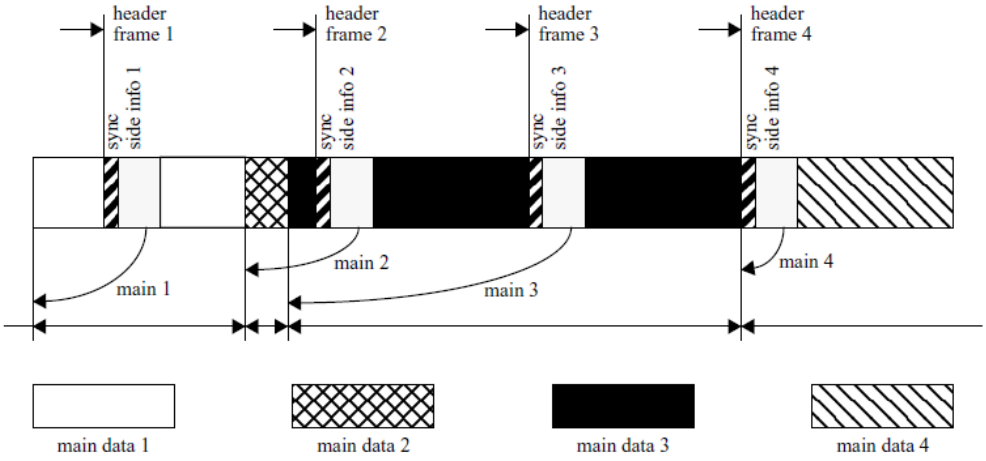
Tabulka 11 – Velikost kvantizačního kroku měřítek

count1table_select (1/2)

Tento bit vyjadřuje jakou Huffmanovu tabulku použít pro dekódování bitů v *count1 regionu*. Nenastavený bit znamená tabulka A, nastavený tabulka B. Více v kapitole 3.7.

3.5 Hlavní data

Hlavní data jsou rozdělena do dvou granulí. Každá granule obsahuje měřítka, data zakódovaná Huffmanovým kódem a nepovinně dodatečné bity. Začátek hlavních dat je bezprostředně za okrajovou informací, nebo je díky *main_data_begin posunut* do některého z předchozích fyzických rámců. V druhém případě musí dekodér při čtení hlavních dat přeskočit 4 bajty hlavičky a okrajovou informaci (17, nebo 32 bajtů). Uspořádání dat je znázorněno na obrázku 1. Použití Huffmanova kódování je hlavním důvodem vysoké komprese MP3 formátu, nižší vrstvy MPEG-1 (layer 1 a 2) toto kódování nepoužívají a takových kompresí nedosahují. Huffmanovo kódování, jakožto bezeztrátovou kompresi dat, používá například i populární kompresní metoda ZIP. Pokud se tedy MP3 soubor zkomprimuje metodou ZIP, nedocílí se žádné dodatečné komprese, naopak velikost souboru může ještě vzrůst o režijní bity ZIP komprese.



Obrázek 7 - Poloha hlavních dat vůči fyzickému rámci (zdroj: [8])

Obrázek 7 ilustruje, že hlavní data nemusí nutně ležet ihned za dodatečnou informací. V tomto případě používá rámec 1 bity z rámce 0, rámec 2 používá bity z rámce 1. V rámci 3 jsou vysoké nároky na zakódování informace, takže používá bity z rámců 1, 2 a 3. Nakonec můžeme vidět rámec 4, který používá pouze bity z rámce 4.

3.6 Získání měřítek

První granule obsahuje měřítka v každém případě, druhá granule je obsahuje v závislosti na *scfsi* (viz 3.4.1). Pokud je nastaven bit pro sdílení měřítek, druhá granule sdílí měřítka pro odpovídající skupinu subpásem společně s první granulí. Jestliže jedna z granulí obsahuje krátká okna, měřítka se nikdy nesdílí.

Počet bitů použitých pro zakódování měřítek se nazývá *part2_length* a vypočítá se podle následujících vztahů v závislosti na oknech použitých v jednotlivých granulích.

Pokud *block_type* = {0,1,3}

$$part2\_length = 11 * slen1 + 10 * slen2 \tag{3.3}$$

Pokud *block_type* = 2 a *mixed_blockflag* = 0

$$part2\_length = 18 * slen1 + 18 * slen2 \tag{3.4}$$

Pokud *block_type* = 2 a *mixed_blockflag* = 1

$$part2\_length = 17 * slen1 + 18 * slen2 \tag{3.5}$$

Připomeňme, že hodnoty slen1 a slen2 mohou být i nulové, tím pádem granule nemusí obsahovat žádná měřítka (uplatní se pouze *global_gain*). Získaná měřítka jsou následně použita v rekvantizačním vztahu, více v 3.8.

3.7 Dekódování Huffmanových bitů

Každý logický rámec může mít až 4 granule – dvě granule na kanál. Každá granule obsahuje 576 zakódovaných frekvenčních vzorků. Pro vzorkovací frekvenci 44100 Hz, vzorek s indexem 0 reprezentuje frekvence okolo 0 Hz a poslední vzorek s indexem 575 reprezentuje frekvence okolo 22050 Hz.

Tyto vzorky jsou rozděleny do pěti oblastí s proměnnou délkou, viz obrázek 6. První tři oblasti jsou nazývány *big_values*, čtvrtá oblast se nazývá *count1 region* (případně *quad region*) a pátá oblast je známá jako *rzero region*. Hodnoty v poslední oblasti jsou všechny nulové, takže vlastně nejsou zakódovány Huffmanovým kódem. Pokud například pomocí hodnot z *big_values* a *count1 regionu* dekódujeme pouze 500 vzorků frekvenčního spektra, zbytek do 576, čili 76 vzorků se doplní nulami a to je právě *rzero region*.

Tři regiony, které patří do *big_values* reprezentují důležité nízké frekvence obsažené v audio signálu. Název *big_values* odpovídá absolutní maximální velikosti, kterou může dekódovaná hodnota mít. Vzorky z této oblasti mohou před rekvantizací nabývat hodnot -8206 až 8206. Pro dekódování hodnot v *big_values* jsou použity tři tabulky definované v normě. Jejich výběr závisí na hodnotě *table_select* z dodatečné informace. Standard pro tuto oblast definuje 16 tabulek spárovaných s dalším parametrem. Tím je dosaženo 30 možností jak hodnoty z této oblasti zakódovat. Po nalezení odpovídajícího Huffmanova kódu z *big_values* dostaneme z tabulky vždy 2 dekódované hodnoty. To je důvod, proč je počet dekódovaných spektrálních čar z *big_values* na obrázku 6 vynásoben dvěma.

Pro příklad je zde uvedena Huffmanova tabulka č. 1. Další tabulky jsou méně, či více obsáhlé. Jejich kompletní přehled je [1], nebo v [7].

kódové slovo	*hodnota*
1	(0, 0)
001	(0, 1)
01	(1, 0)
000	(1, 1)

Tabulka 12 - Huffmanova tabulka č. 1

Dekódování hodnot z *big_values* pomocí tabulky 1 probíhá následovně. Mějme například hlavní data obsahující tyto bity 000101010 atd. Nejprve musíme dekódovat bity jako běžné řetězce zakódované Huffmanovým kódem. První 3 bity „000“ odpovídají dvěma výstupním vzorkům 1 a 1. Říkejme jim *x* a *y*. Pokud jsou dekódované hodnoty různé od nuly, jako v našem případě, čteme ještě postupně jeden bit pro *x* a jeden bit pro *y*. Tyto bity mají význam znaménka. Pokud je bit nastaven, dekódovaná hodnota má opačné znaménko. V našem případě je znaménkový bit pro *x* roven 1 a pro *y* roven 0, takže výsledkem dekódování jsou 2 hodnoty -1 a 1.

Žádná z tabulek definovaných ve standardu nemá dekódovanou hodnotu vyšší, než 15. Přesto, jak již bylo zmíněno, může maximální dekódovaná hodnota z *big_values* nabývat v absolutní hodnotě velikost až 8206. Pokud bychom chtěli hodnoty dekódovat jen jako v našem příkladě, Huffmanovy tabulky by byly obrovské. Proto se společně s Huffmanovou tabulkou udává ještě další parametr, který se nazývá *linbits*. Ne všechny tabulky používají *linbits*, respektive mají tento parametr nulový, jako již dříve zmíněná tabulka č. 1. Pokud je dekódovaná hodnota maximální, čili 15, a tabulka má *linbits* větší než 0, přičteme z granule právě tolik dalších bitů, jako je velikost *linbits* a tuto hodnotu přičteme k dekódované hodnotě. Jelikož *linbits* je maximálně 13 (pro tabulky 23 a 31), může být takto přičtená hodnota rovna 2^{13}, což je 8191. Pokud sečteme 15 a 8191, dostaneme již několikrát zmíněnou maximální hodnotu 8206. I v případě tabulek, které mají definovány *linbits* se čtou znaménkové bity. *Linbits* jsou důvodem toho, pro stačí jen 16 různých tabulek a přitom máme 30 možností jak hodnoty zakódovat. Oblast s nižší dynamikou si vystačí s tabulkou, která má nižší *linbits* a tím pádem se ušetří nemalé množství bitů.

Algoritmus dekódování hodnot z oblasti *big_values* je následující:

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*.

V *count1 regionu* jsou zakódovány vysoké frekvence, které většinou nemají vysoké amplitudy, takže mohou být zakódovány velice úsporně. Všechny dekódované hodnoty z této oblasti jsou v rozmezí <-1;1>. Pro tuto oblast se používají pouze dvě Huffmanovy tabulky, nazývané A a B, odlišné od tabulek pro *big_values*, kompletní obsahy těchto tabulek je uveden v [7]. Tyto dvě tabulky nepoužívají *linbits* a jejich výstupem jsou hned 4 dekódované hodnoty najednou. Při dekódování se stále používají znaménkové bity, jako v minulém případě.

Algoritmus dekódování hodnot z oblasti *count1* je následující:

1. Dekódujeme bity podle Huffmanovy tabulky A nebo B v závislosti na hodnotě *count1table_select*. Hodnoty nazveme *v*, *w*, *x* a *y*.
2. Pokud *v* není 0, přečteme znaménkový bit. Pokud je bit nastaven, $v = -v$.
3. Opakujeme krok 2 pro *w*, *x* a *y*.

Dekódování podle tabulek provádíme pouze do vyčerpání bitů určených hodnotou *part2_3_length*. Pokud jsme všechny bity vyčerpali a ještě nemáme 576 frekvenčních čar, zbylé doplníme nulami a vznikne tím *rzero region*.

Po získání spektrálních složek signálu můžeme přistoupit k rekvantizaci.

3.8 Rekvantizace

Rekvantizace je dalším klíčovým krokem při dekódování MP3 formátu. Důvodem kvantizace prováděné v enkodéru je nutnost upravit vzorky spektra tak, aby se daly zakódovat huffmanovým kódem. Jejich velikost je totiž omezená a *global_gain* pomáhá snížit velikosti vzorků do mezí vhodných k zakódování. Dalším důvodem je potlačení kvantovacího šumu. K tomu slouží měřítka. Pásmo od nultého do 575. vzorku je rozděleno na 21, respektive 12 měřítkových subpásem a v každém takovém subpásmu může být použito jiné měřítko. Tato technika umožňuje s malou chybou nakvantovat i signály, které mají velice odlišné amplitudy frekvenčních vzorků v různých subpásmech. Signály s nízkou amplitudou jsou díky tomu mnohem méně postižené kvantovacím šumem.

Obrázek 8 - Rozdělení pásma na subpásma měřítek - dlouhý blok 44100 Hz (zdroj: [9])

Obrázek 8 ilustruje rozdělení frekvenčního pásma na části pro aplikaci různých měřítek. Toto rozdělení je různé pro každou vzorkovací frekvenci a pro typ bloku (dlouhý, krátký, nebo smíšený). Hranice jednotlivých pásem jsou určeny tabulkou v normě [1],[7]. Po rozdělení do subpásem je postupně na každý frekvenční vzorek aplikován rekvantizační

vztah. Hodnota dekódovaná z Huffmanových bitů je označena jako *is* a rekvantizovaná hodnota je označena *xr*.

Rekvantizační vztahy pro jednotlivé typy bloků pro i od 0 do 575.

Krátké bloky:

$$xr[i] = sign(is[i]) * |is[i]|^{\frac{4}{3}} * 2^{A} * 2^{B}, \tag{3.6}$$

kde pro A a B platí:

$$A = \frac{1}{4} * (global\_gain[gr] - 210 - 8 * subblock\_gain[okno][gr]), \tag{3.7}$$

$$B = -(scalefac\_scale * scalefac\_s[gr][ch][sfb][okno]. \tag{3.8}$$

Dlouhé bloky:

$$xr[i] = sign(is[i]) * |is[i]|^{\frac{4}{3}} * 2^{A} * 2^{B}, \tag{3.9}$$

kde pro A a B platí:

$$A = \frac{1}{4} * (global\_gain[gr] - 210), \tag{3.10}$$

$$B = -scalefac\_scale * (scalefac\_l[sfb][ch][gr][okno] + preflag[gr] * pretab[sfb]). \tag{3.11}$$

Zopakujme, že vztah pro dlouhé bloky se používá při *block_type* = {0,1,3} a vztah pro krátké bloky při *block_type* = 2. Pokud je mezi nynějším a předchozím časovým rámcem velmi malý rozdíl, enkodér použije dlouhé (normální) okno. V opačném případě použije enkodér krátké okno pro dosažení lepšího časového rozlišení MDCT [9].

3.9 Přerovnání spektra

Přerovnání spektra je nutné pro zajištění správného vstupu do IMDCT, ale pouze pro krátké bloky, tzn. *block_type* = 2. Přerovnání krátkých bloků je provedeno enkodérem z toho důvodu, aby se zajistila větší efektivita Huffmanova kódu. Krátká okna jsou před přerovnáním seřazena nejdříve podle měřítkových subpásem, potom podle okna a nakonec podle frekvence. Oproti tomu dlouhá okna jsou seřazena nejdříve podle měřítkových subpásem a potom podle frekvence [7].

Prokládání vzorků krátkých oken nejlépe vysvětlí příklad. Pokud má první měřítkové subpásmo velikost 4, toto jsou dekódovaná data:

a[0], b[0], c[0], a[1], b[1], c[1], a[2], b[2], c[2], a[3], b[3], c[3], a[4], atd.,

přerovnaná data vypadají potom takto:

a[0], a[1], a[2], a[3], b[0], b[1], b[2], b[3], c[0], c[1], c[2], c[3], a[4], atd.,
kde a, b, c jsou krátká okna o 192 vzorcích.

3.10 Zpracování sterea

Pokud má dekódovaný signál 2 kanály zakódované metodou joint stereo, je nutné rekonstruovat vzorky levého a pravého kanálu podle metody uvedené v hlavičce.

Dva kanály běžného stereo signálu nejsou nezávislé a joint stereo se snaží využít této závislosti k omezení datového toku. Zpracování joint sterea je velice komplikované, protože krátké bloky jsou zpracovány odlišně od dlouhých bloků. Granule může obsahovat i kombinaci dlouhých a krátkých oken a jednotlivá subpásma mohou obsahovat jinou metodu kódování joint sterea. Jak již bylo zmíněno v 3.2.2, existují dva typy joint sterea.

3.10.1 M/S stereo

Middle/side stereo, čili přenos součtového a rozdílového kanálu je velice jednoduchou formou bezztrátové komprese stereo signálu. Součtový kanál, jak už název napovídá, je součtem levého a pravého kanálu a rozdílový kanál je jejich rozdílem. Rozdíly mezi oběma kanály bývají malé, takže rozdílový kanál neobsahuje mnoho informací a dá se efektivně zakódovat. Rekonstrukce vzorků pravého a levého kanálu probíhá podle vztahů 3.12 a 3.13.

$$L(i) = \frac{1}{\sqrt{2}} * [M(i) + S(i)] \tag{3.12}$$

$$R(i) = \frac{1}{\sqrt{2}} * [M(i) - S(i)] \tag{3.13}$$

Kde L(i) a R(i) jsou rekonstruované vzorky levého, respektive pravého kanálu, M(i) jsou vzorky součtového a S(i) jsou vzorky rozdílového kanálu. Jedinou zvláštností je dělení odmocninou ze 2, nikoliv pouze dvěma. To je z důvodu zefektivnění kvantizace enkodérem [8].

3.10.2 Intensity stereo

V tomto módu enkodér zakóduje stereo signál pouze do jednoho součtového kanálu a místo měřítek pro pravý kanál se přenáší speciální proměnná $ispos_{sfb}$, která určuje váhu (poměr hlasitosti levého a pravého kanálu).

Dekodér rekonstruuje levý a pravý kanál pouze z jednoho součtového signálu (L'$_i$), který je přenášen v levém kanálu a váhy, která je přenášena namísto měřítek pravého kanálu. Rekonstrukce kanálů se řídí podle následujících vztahů [8]:

$$isratio_{sfb} = tan\left(ispos_{sfb} * \frac{\pi}{12}\right), \tag{3.14}$$

$$L_i = L'_i * \frac{isratio_{sfb}}{1+isratio_{sfb}}, \tag{3.15}$$

$$R_i = L'_i * \frac{1}{1+isratio_{sfb}}. \tag{3.16}$$

Parametr $ispos_{sfb}$ může nabývat pouze hodnot 0 až 6, takže funkce tangens může být z důvodu urychlení výpočtu nahrazena jednoduchou tabulkou.

3.11 Redukce aliasů

V dlouhých blocích je před vstupem do IMDCT potřeba redukovat aliasing, který vznikl v enkodéru díky polyfázovým filtrům. Redukce je docíleno pomocí výpočtu osmi motýlků pro každé subpásmo s koeficienty *cs* a *ca* vypočtenými podle 3.17 a 3.18.

$$cs(i) = \frac{1}{\sqrt{(1+c(i))^2}} \tag{3.17}$$

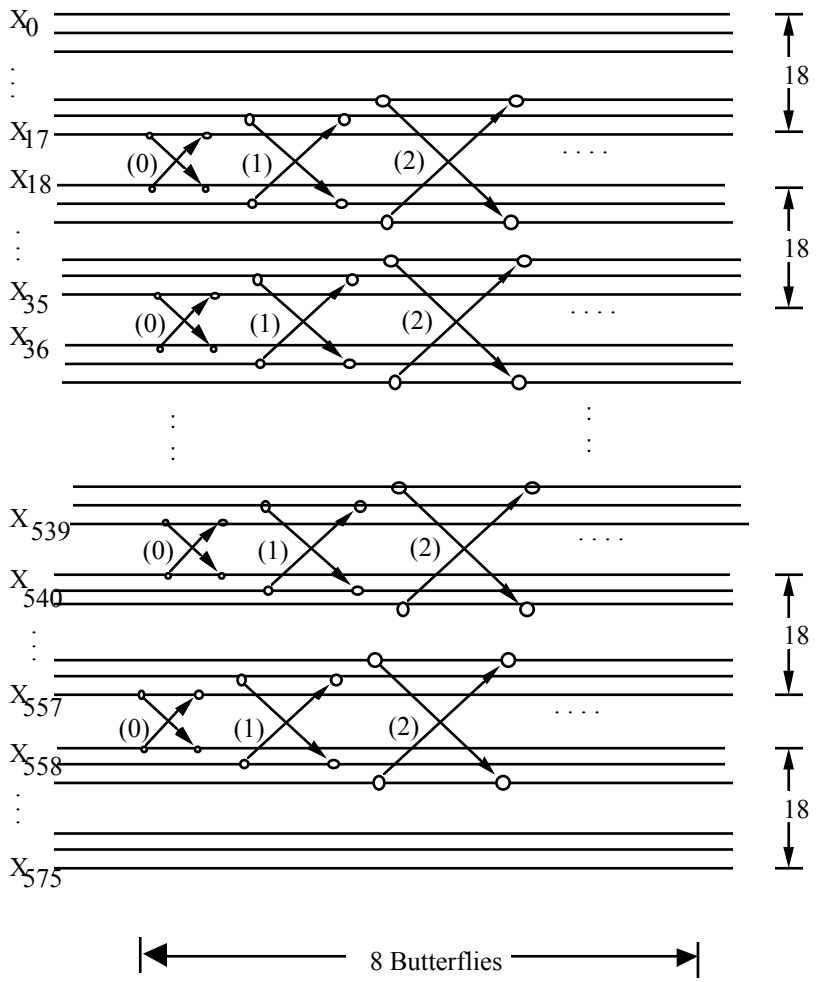
$$ca(i) = \frac{c(i)}{\sqrt{(1+c(i))^2}} \tag{3.18}$$

Hodnota c(i) je dána pro všech 8 motýlků tabulkou.

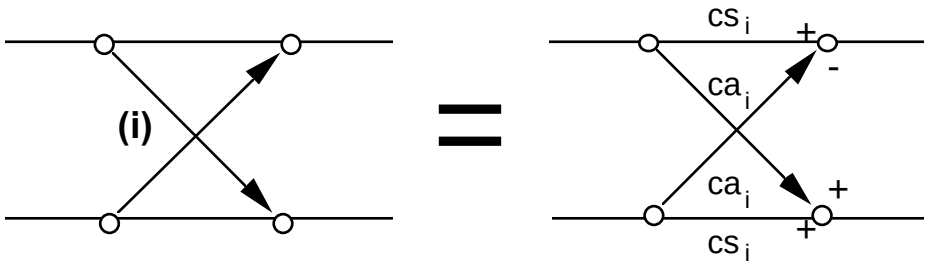
i	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***
c(i)	-0,6	-0,535	-0,33	-0,185	-0,095	-0,041	-0,00142	-0,0037

Obrázek 9 - Hodnoty c(i) pro redukci aliasů

Celý proces počítání motýlků ilustrují obrázky10 a 11. Pro krátké bloky není redukce aliasů potřebná.



Obrázek 10 - Schéma výpočtu pro redukci aliasů v celém rozsahu spektra (zdroj: [7])



Obrázek 11 - Jeden motýlek redukce aliasů (zdroj: [7])

3.12 Transformace do časové oblasti

Konverze 576 získaných frekvenčních složek do časové oblasti se řídí podle hodnot parametrů *block_type* a *mixedblock_flag*. Pro pochopení významu těchto parametrů je dobré si nejprve vysvětlit funkci enkodéru.

3.12.1 Enkodér – banka filtrů a transformace

Do enkodéru vstupují časové vzorky nejčastěji ve formátu 16 bit PCM, protože tento formát používá dobře známé hudební CD. Enkodér vezme každých 576 časových vzorků a zakóduje je do granule. Každý rámec obsahuje 2 granule na kanál. Enkodér rovněž uloží do hlavičky a dodatečné informace, jakým způsobem byl daný signál zakódován.

Vzorky z časové oblasti jsou transformovány do frekvenční oblasti v několika krocích, které se aplikují vždy na jednu granuli, dokud nedojde k dokončení transformace. Potom se transformuje další granule.

3.12.1.1 Analyzující banka polyfázových filtrů

Nejprve je 576 časových vzorků přivedeno na vstup sady 32 pásmových propustí, z nich každá vyprodukuje 18 časových vzorků reprezentujících 1/32tinu frekvenčního spektra. Pokud je vzorkovací frekvence 44100 Hz, každé pásmo je široké cca 689 Hz. Je třeba upozornit na to, že zde dochází k podvzorkování. Výstup běžné pásmové propusti pro 576 vstupních vzorků je opět 576 vzorků, nicméně filtry používané při enkódování MP3 výstupní signál decimují a výstupem je jen každý 32. vzorek. Díky tomu je celkový počet výstupů ze všech filtrů stejný, jako počet vstupních vzorků.

Tato část kodéru je známá jako analyzující banka polyfázových filtrů. Dekodér ke konci dekódujícího procesu použije opačný proces, kdy zkombinuje takto získaná subpásma do originálního signálu. Tento opačný proces se nazývá syntéza pomocí banky polyfázových filtrů. Tyto dvě zmíněné verze bank filtrů jsou principiálně jednoduché, ale matematicky velice složité.

3.12.1.2 MDCT

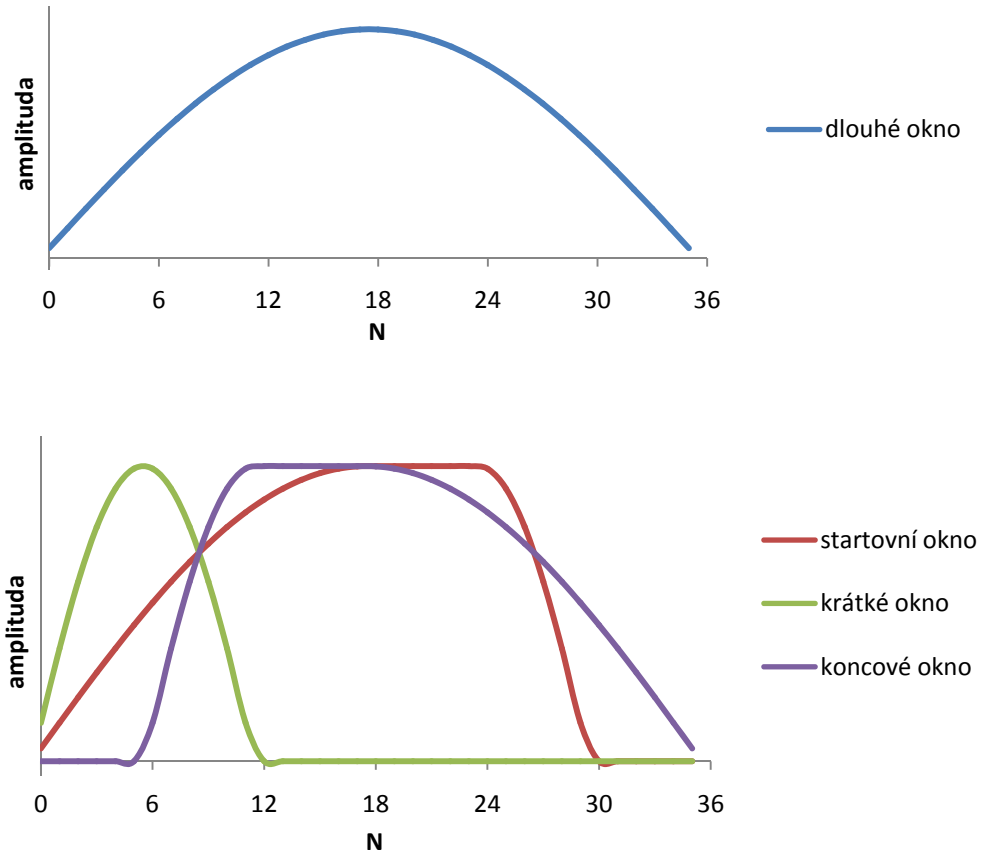
Výstup z každého pásmového filtru je transformován pomocí MDCT. Toto je klíčová činnost enkódování MPEG-1 layer 3, protože zajišťuje dobré frekvenční rozlišení.

MDCT vezme signál a vyjádří ho jako součet kosinových složek ve frekvenční oblasti. V porovnání s FFT, DCT a ostatními dobře známými transformacemi, má MDCT několik vlastností, které ji předurčují k použití při kompresi audio signálů.

Za prvé, MDCT má tu vlastnost, že je energeticky kompaktní, podobně jako některé další kosinové transformace. To znamená, že nejvíce informace je koncentrováno v několika málo výstupních vzorcích s velkou energií. To znamená, že pokud na vstupním signálu provedeme (M)DCT a všechny „malé“ výstupní frekvenční složky vynulujeme a poté provedeme zpětnou transformaci, výsledkem jsou pouze malé změny oproti originálnímu vstupnímu signálu. Tato vlastnost je velice užitečná a proto se kosinové transformace používají i pro kódování obrázků (JPEG) a videa.

Za druhé, MDCT je navržena na zpracovávání posloupnosti bloků dat, takže má, v porovnání s ostatními transformacemi, menší nesrovnalosti na okrajích bloků. MDCT je někdy také nazývána jako „překrývaná“ transformace. To znamená, že při zpracování

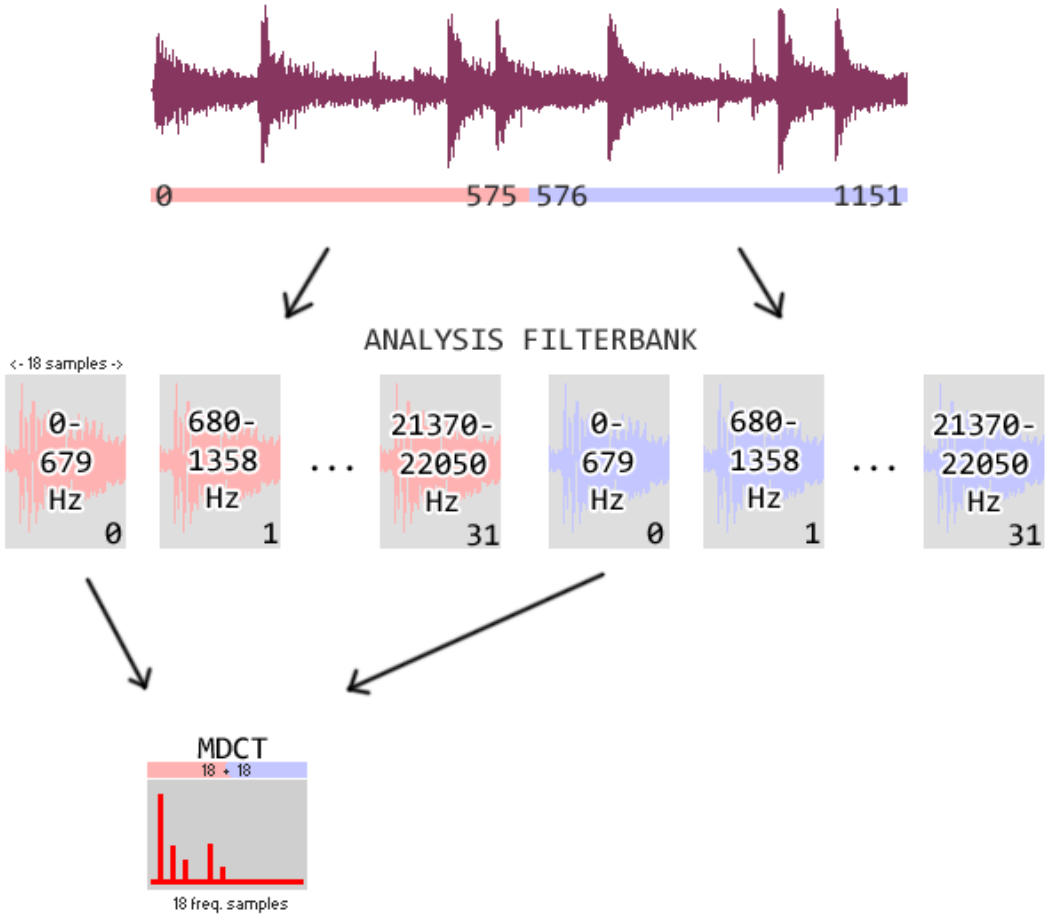
aktuálního bloku používáme i data z minulého bloku. Vstupem je 2N časových vzorků a výstupem je N frekvenčních vzorků. Místo samostatného transformování bloků dlouhých 2N jsou sousední bloky překrývány. Toto překrývání pomáhá redukovat artefakty na rozhraní bloků. Enkodér nejprve provede MDCT, řekněme na vzorcích 0-35, potom 18-53, 36-71 atd. Pro další vyhlazení hranic jednotlivých bloků se používá váhování oknem, které je prováděno ještě před MDCT. Enkodér uloží informaci o použitém váhovacím okně do *block_type*. Tvary váhovacích oken jsou zobrazeny na obrázku 12. Jejich rovnice jsou uvedeny dále v kapitole 3.12.2.



Obrázek 12 - Typy váhovacích oken

V případě MP3 se MDCT provádí na subpásmech z analyzující banky filtrů. Aby bylo dosaženo všech dobrých vlastností MDCT, transformace není prováděna přímo na 18 vzorcích, ale na oknem váhovaném signálu, který vznikne spojením 18 minulých a současných vzorků. Transformace prvního subpásma je vysvětlena na obrázku 13, kde jsou zobrazeny 2 po sobě následující granule. Připomeňme, že se jedná o pohled na funkci enkodéru.

MDCT může být použita buď na 36 vzorků, jak bylo popsáno výše, nebo jsou provedeny 3 MDCT na každých 12 vzorků. První volba se nazývá dlouhá metoda a dává nám lepší frekvenční rozlišení. Druhá metoda je známá jako krátká a poskytuje lepší časové rozlišení. Enkodér zvolí dlouhou MDCT pro dosažení lepší kvality, pokud se signál mění jen velice málo a krátkou metodu, pokud je v signálu hodně změn.



Obrázek 13 - MDCT prvního subpásma (zdroj: [9])

Pro celou granuli (576 vzorků) může enkodér provést dlouhou MDCT na všech subpásmech – výsledek se nazývá dlouhý blok, nebo může provést krátkou MDCT ve všech subpásmech – tím vznikne krátký blok. Je zde i třetí možnost nazývaná smíšený blok. V posledním případě použije enkodér dlouhou MDCT na první 2 subpásma a na zbytek použije krátkou MDCT. Tato možnost se používá v případě, kdy je nutné dobré časové rozlišení, ale krátká MDCT na celém pásmu by způsobovala artefakty. První 2 subpásma jsou zpracována dlouhou MDCT, protože lidské ucho je v tomto pásmu nejcitlivější (pro vzorkovací frekvenci 44100 Hz to odpovídá pásmu 0 až cca 1,4 kHz). Hranice použití dlouhé a krátké MDCT je v MP3 pevná, což je často kritizovaný nedostatek návrhu komprese MPEG-1 layer3. Je třeba zmínit, že ne všechny enkodéry podporují smíšené bloky.

Kombinace analyzující banky filtrů a MDCT je známá pod názvem hybridní analyzující banka filtrů a je to velice matoucí část enkodéru. Dekodér obsahuje blok, který má opačnou funkci a nazývá se hybridní syntetizující banka filtrů. Stojí za to zmínit, že novější audio kodek definovaný v MPEG-2 (AAC) je v této oblasti jednodušší díky absenci pásmových filtrů. V AAC je prováděna MDCT přímo z celé granule.

3.12.2 Dekodér

3.12.2.1 IMDCT

Díky znalosti fungování enkodéru docházíme k překvapivému závěru, že právě překrývání MDCT je důvod, proč nemůžeme dekódovat rámce, ani granule odděleně. Díky vlastnostem použité kosinové transformace potřebujeme pro inverzní transformaci aktuálního bloku dat výstupní data z minulého bloku.

Toto je důvod parametrů *block_type* a *mixed_blockflag*. V případě krátkého bloku použije dekodér 36bodovou IMDCT a překrývání s minulou granulí na všech 32 subpásem. Pokud se jedná o krátký blok, jsou použity tři 12bodové IMDCT. Ještě je zde možnost smíšeného bloku, kde je na dolní 2 subpásma použita dlouhá IMDCT a na další subpásma krátká transformace.

Hodnota *block_type* také určuje, jaké váhovací okno se použije po inverzní transformaci. Znázornění jednotlivých oken je na obrázku 12. Vztah 3.19 popisuje váhovací okno dlouhého bloku. Jedná se vlastně o první půlperiodu funkce sinus. Vztah 3.20 vyjadřuje startovní okno. Vztah 3.21 popisuje krátké okno a nakonec 3.22 vyjadřuje koncové okno.

$$w_{long}(i) = \sin\left[\frac{\pi}{36} * \left(i + \frac{1}{2}\right)\right] \qquad \text{pro } i \in \langle 0; 35\rangle \tag{3.19}$$

$$\begin{aligned} w_{start}(i) &= \sin\left[\frac{\pi}{36} * \left(i + \frac{1}{2}\right)\right] && \text{pro } i \in \langle 0; 17\rangle \\ w_{start}(i) &= 1 && \text{pro } i \in \langle 18; 23\rangle \\ w_{start}(i) &= \sin\left[\frac{\pi}{12} * \left(i + \frac{1}{2} - 18\right)\right] && \text{pro } i \in \langle 24; 29\rangle \\ w_{start}(i) &= 0 && \text{pro } i \in \langle 18; 23\rangle \end{aligned} \tag{3.20}$$

$$\begin{aligned} w_{short}(i) &= \sin\left[\frac{\pi}{12} * \left(i + \frac{1}{2}\right)\right] && \text{pro } i \in \langle 0; 11\rangle \\ w_{short}(i) &= 0 && \text{pro } i \in \langle 12; 35\rangle \end{aligned} \tag{3.21}$$

$$
\begin{aligned}
w_{end}(i) &= 0 && \text{pro } i \in \langle 0;5\rangle \\
w_{end}(i) &= \sin\left[\frac{\pi}{12} * \left(i + \frac{1}{2} - 6\right)\right] && \text{pro } i \in \langle 6;11\rangle \\
w_{end}(i) &= 1 && \text{pro } i \in \langle 12;17\rangle \\
w_{end}(i) &= \sin\left[\frac{\pi}{36} * \left(i + \frac{1}{2}\right)\right] && \text{pro } i \in \langle 18;35\rangle
\end{aligned}
\tag{3.22}
$$

Startovací a koncové okno se používá v případě, kdy mezi granulemi dochází ke změně z dlouhého na krátké, respektive z krátkého na dlouhé okno.

IMDCT transformuje frekvenční čáry *Xk* na vzorky subpásem polyfázového filtru. Vztah 3.23 vyjadřuje výpočet IMDCT, kde n je 12 pro krátké bloky a 36 pro dlouhé bloky.

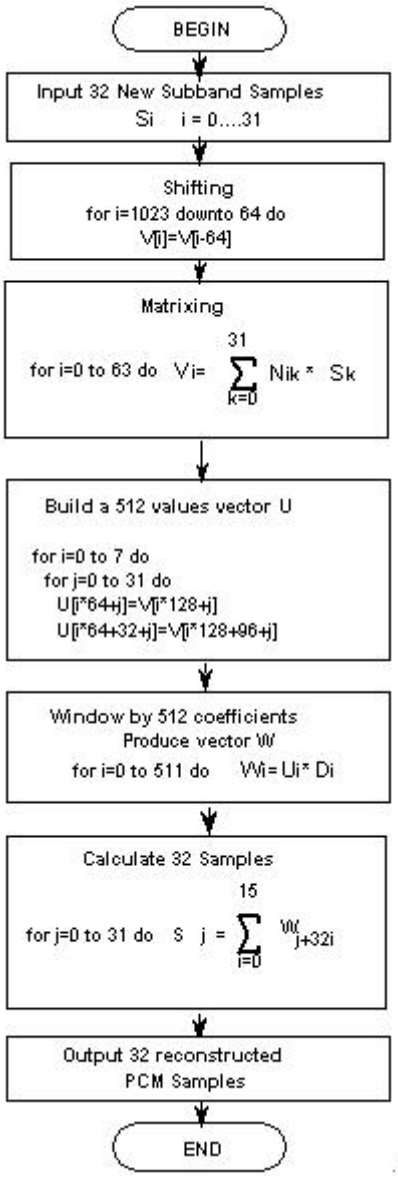
$$
x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left[\frac{\pi}{2n}\left(2i + 1 + \frac{n}{2}\right)(2k+1)\right] \qquad \text{pro } i \in \langle 0; n-1 \tag{3.23}
$$

3.12.2.2 Frekvenční inverze

Za účelem vyrovnání frekvenční inverze v syntezující bance polyfázových filtrů je každý lichý vzorek každého lichého subpásma vynásoben hodnotou -1.

3.12.2.3 Syntetizující banka polyfázových filtrů

Syntetizující banka polyfázových filtrů transformuje 32 subpásem po 18 časových vzorcích v každé granuli na 18 bloků po 32 PCM vzorcích. Tato operace je posledním krokem k úplnému dekódování MP3 souboru do podoby PCM vzorků. Banka filtrů pracuje s 32 vstupními vzorky najednou, každý z jednoho subpásma.



Obrázek 14 - Algoritmus syntézy časových vzorků na výstupní PCM signál (zdroj: [11])

Algoritmus syntézy je zřejmý z obrázku 14. Prvním krokem syntézy je transformování 32 vzorků (postupně jeden z každého subpásma) na 64 hodnot. Výsledkem je tzv. *V* vektor. K transformaci se používá varianta MDCT nazývaná „matrixing“. *V* vektor je následně umístěn do FIFO fronty, která uchovává posledních 16 *V* vektorů, to znamená, že velikost této fronty je 1024 vzorků. Tuto frontu je nutné vždy vyprázdnit na začátku dekódování a při posouvání pozice přehrávání.

Výpočet vektoru *V* probíhá následovně:

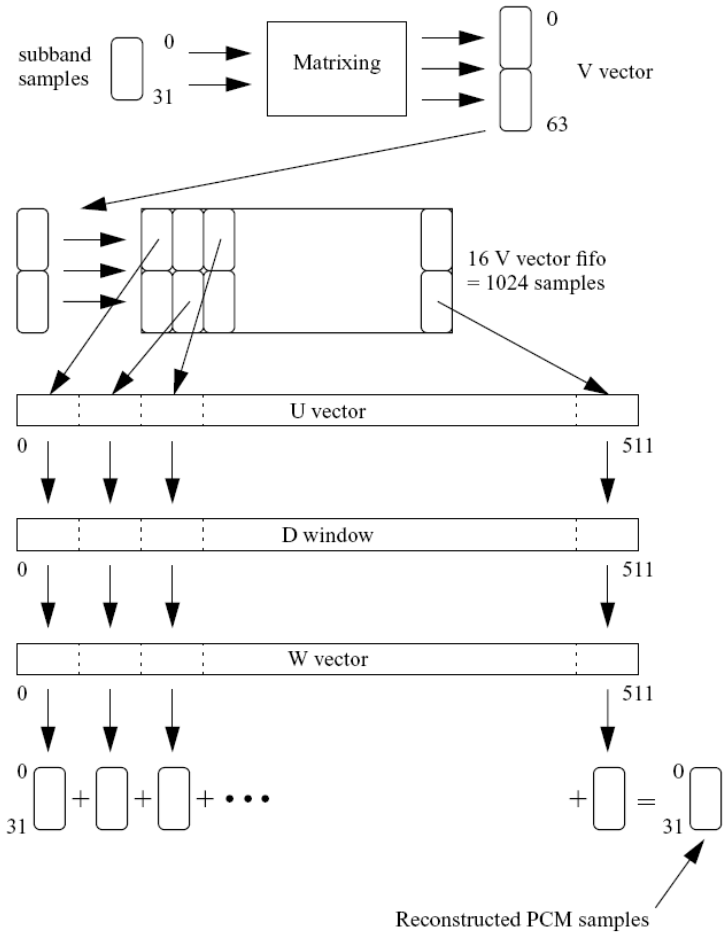
pro i = 0 do 63 provést:

$$V_i = \sum_{k=0}^{31} N_{ik} * S_k, \tag{3.24}$$

kde:

$$N_{ik} = \cos\left[(16 + i)(2k + 1)\frac{\pi}{64}\right] \tag{3.25}$$

a S_k jsou vzorky ze subpásem.



Obrázek 15 - Syntezující banka polyfázových filtrů (zdroj: [8])

Následně je vytvořen *U* vektor střídavým vybíráním 32 vzorků dlouhých bloků z FIFO fronty, jak je zobrazeno na obrázku 15. Na vektor *U* je aplikováno váhování oknem, tím vznikne vektor *W*. Koeficienty okna jsou dány tabulkou. Rekonstruované vzorky jsou získány opětovnou dekompozicí vektoru *W* na 18 vektorů, každý s 32 vzorky, a následným sečtením těchto vektorů. Tímto končí proces dekódování a výstupní vzorky je možné přehrát.

4. DSP procesory řady SHARC

Dekodér bude naprogramován jako softwarový modul pro jeden z DSP procesorů řady SHARC od firmy Analog Devices. SHARC je zkratkou pro Super Harvard Architecture, tedy česky super harvardská architektura. Slovo super značí vylepšení oproti klasické harvardské architektuře. Tímto vylepšením je použití instrukční cache. Jedná se o vysoce výkonné DSP s celočíselnou aritmetikou a aritmetikou s plovoucí řádovou čárkou. Procesory SHARC mají široké využití v oblasti digitálního zpracování signálu, hlavně díky nízké spotřebě vzhledem k jejich výkonu.

V současné době jsou dostupné procesory SHARC čtyř generací. Můžeme si vybrat od nejlevnějších modelů, až po nejvýkonnější modely, které poskytují výpočetní výkon v plovoucí a pevné řádové čárce až 2700 MFLOPs při taktovací frekvenci 450 MHz.

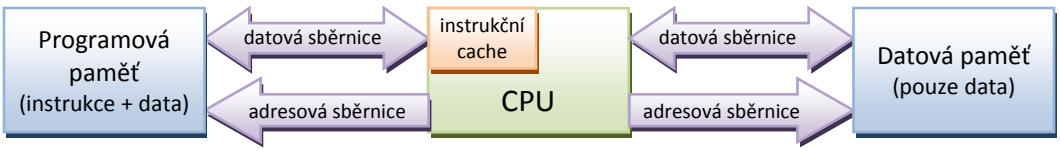
4.1 Architektura SHARC

Procesory SHARC jsou 32 bitové tzv. VLIW procesory. To znamená, že díky délce instrukčního slova 48 bitů, může procesor provádět několik paralelních operací najednou. Ne však libovolné kombinace. Blokové schéma architektury SHARC můžeme vidět na obrázku 16.

Instrukční cache procesoru SHARC umožňuje využít 2 oddělené dvouportové paměti a paměťové sběrnice pro načítání dvou různých dat v jednom okamžiku. Tato vlastnost je výhodná při většině operací číslicového zpracování signálu, jako je konvoluce, korelace, výpočty FIR filtrů atd.

Společné vlastnosti všech generací procesorů této architektury:

- 32/40bitová IEEE aritmetika s plovoucí řádovou čárkou,
- 32bitové násobičky s pevnou řádovou čárkou s 64bitovým výsledkem a 80bitovým akumulátorem,
- žádná aritmetická pipeline - všechny výpočty jsou dlouhé jeden instrukční cyklus,
- podpora kruhového adresování přímo v hardware,
- 32 adresových ukazatelů pro 32 kruhových bufferů,
- až 6 úrovní vnořených hardwarových smyček s nulovou režií,
- instrukční sada podporuje podmíněnou aritmetiku, manipulaci s bity, dělení, druhou odmocninu a bitové operace,
- DMA umožňující datové přenosy na pozadí s nulovou režií na plné taktovací frekvence bez účasti procesoru,
- vysokorychlostní sériová rozhraní pro připojení periférií.



Obrázek 16 - Super harvardská architektura

4.2 Generace procesorů SHARC

4.2.1 První generace

První generace procesorů SHARC nabízí výkon až 198 MFLOPs na 66 MHz [12], který je v porovnání s novými generacemi sice velice malý, ale přesto dostatečný pro nasazení ve spotřební elektronice, čemuž nahrává i nízká cena.

Vlastnosti ADSP-21065L

- ADSP-21065L nabízí výpočetní výkon 198 MFLOPs, nebo 198 MOPs,
- integrovaná 16k x 32bitová dvouportová paměť,
- I2S sběrnice s až 8 kanály,
- dva vysokorychlostní sériové porty s podporou 32kanálového TDM,
- dva časovače pro zachycení událostí a PWM funkce,
- 12 programovatelných I/O pinů,
- 10 DMA kanálů.

4.2.2 Druhá generace

Druhá generace přinesla zdvojnásobení výkonu v oblasti zpracování signálu (600 MFLOPs na 100 MHz) [12]. Toho bylo docíleno použitím architektury SIMD – jedna instrukce, dvojí data. Toto hardwarové rozšíření první generace SHARC zdvojnásobilo dostupné množství výpočetních prostředků. Procesory druhé generace obsahují dvě násobičky, dvě ALU a dvojí soubor systémových registrů, což v určitých aplikacích přináší zdvojnásobení výkonu. Tato schopnost je užitečná zejména v oblasti zpracování audio signálů, kde mohou být zpracovány oba kanály sterea najednou.

Vlastnosti ADSP-21160N

- Všechny instrukce provedené v jednom instrukčním cyklu včetně SIMD operací v obou výpočetních jednotkách,
- špičkový výkon 570 MFLOPs a udržitelný výkon 380 MFPLOPs (při výpočtu FIR),

- dvojité datové adresové generátory (DAG) s modulo a bitově-reverzibilním adresováním,
- JTAG testovací rozhraní.

4.2.3 Třetí generace

Třetí generace produktů SHARC přináší vylepšenou architekturu SIMD a tím i zvýšený výkon na 2400 MFLOPs při 400 MHz [12]. Tyto produkty obsahují velké množství uživatelsky programovatelné paměti ROM a mnoho periferií využitelných při zpracování audia. Mezi ně patří S/PDIF vysílač/přijímač, 8kanálový asynchronní konvertor vzorkovací frekvence, 4 precizní generátory hodin atd.

Vlastnosti ADSP-21369

- Jádro SHARC běžící na 400 MHz se špičkovým výkonem 2400 MFLOPs,
- 2 Mb SRAM, 6 Mb uživatelsky programovatelné ROM,
- Vysokorychlostní 32bitové rozhraní pro připojení externích pamětí SRAM, SDRAM a flash,
- digitální audio rozhraní (DAI) umožňující uživatelsky definovaný přístup k periferiím včetně osmi sériových portů, S/PDIF, 8kanálovému konvertoru vzorkovací frekvence a čtyř precizních generátorům hodin,
- 34 DMA kanálů s nulovou režií,
- 16 PWM kanálů.

4.2.4 Čtvrtá generace

Čtvrtá generace nabízí opětovné zvýšení výkonu až na 2700 MFLOPs při 450 MHz [12]. Obsahuje hardwarové akcelerátory výpočtu číslicových filtrů a nový způsob konfigurace paměti a nové audio periferie, které podporují nejnovější algoritmy zpracování zvuku. Nejnovější generace také umožňuje připojení rychlejší externí paměti DDR2 SDRAM s šířkou slova 16 bitů.

Vlastnosti ADSP-21469

- Taktovací frekvence 450 MHz,
- 5 Mb RAM na čipu,
- akcelerátory výpočtů FIR, IIR a FFT,
- rozhraní externí paměťové sběrnice pro připojení DDR2 paměti,
- 8 sériových portů s módy TDM a I2S,
- DAI pro uživatelsky definovatelný přístup k periferiím,
- 2 SPI kompatibilní porty podporující master a slave módy.

5. Vývojový kit ADZS-21369 EZ-KIT lite

Pro praktické testy byl vybrán vývojový kit ADZS-21369 EZ-KIT lite. Důvodem byla hlavně dostupnost v době vývojových prací. Nicméně kit disponuje všemi potřebnými periferními obvody, jako jsou např. velká SDRAM pro uložení zdrojového souboru, nebo audio kodek pro zajištění zvukového výstupu.

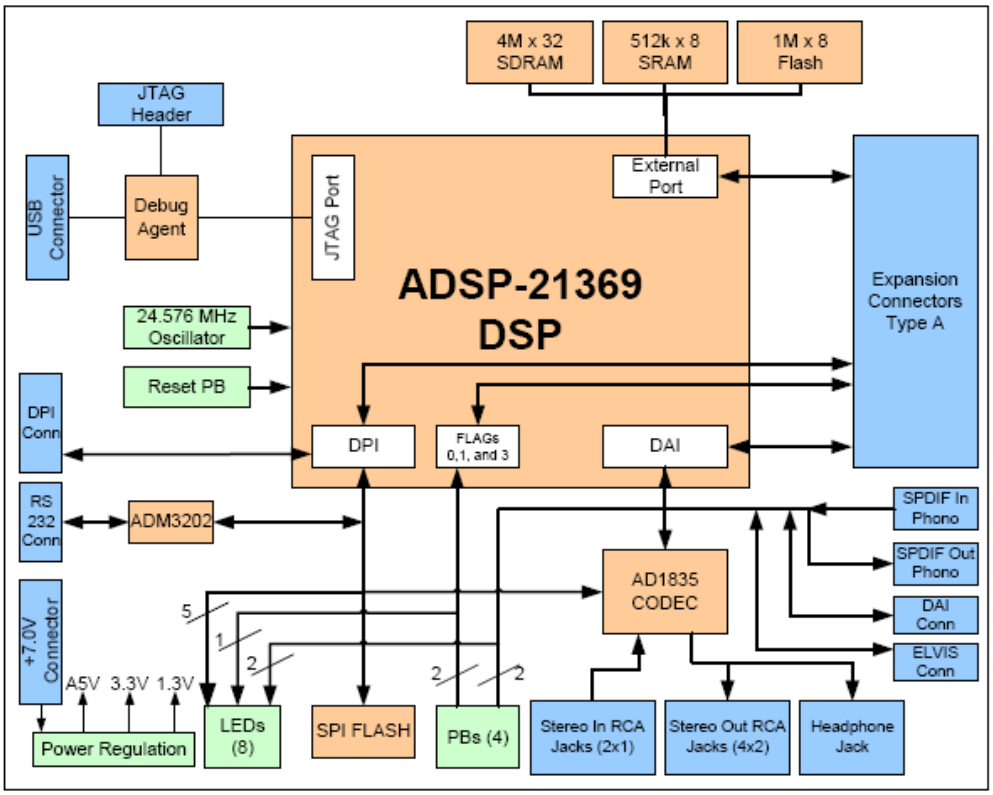
Hardware vývojového kitu je založen na procesoru 3. generace ADSP-21369 s rozšířenou architekturou SIMD. Na vývojovém kitu najdeme 4 druhy pamětí pro různé účely, integrované USB ladicí rozhraní spolupracující s dodávaným software, UART, S/PDIF vstup/výstup, konektor jack 3,5 mm pro připojení sluchátek, 10 konektorů typu cinch (RCA) pro vstupy/výstupy audio kodeku a další rozšiřující konektory.

Obrázek 17 - Vývojový kit ADZS-21369 (zdroj: [13])

Výčet vlastností kitu [13]:

- procesor ADSP-21369 v pouzdře BGA,
- audio kodek AD1835, vzorkovací frekvence 96 kHz, rozlišení až 24 bitů,
- 1 M x 8 b flash paměť,
- 1 M x 32 b čtyřbanková SDRAM,
- 512 k x 8 b SRAM,
- 2 Mb SPI flash paměť,
- ADN3202 RS-232 vysílač/přijímač s rychlostí až 460 kb/s,
- USB ladicí rozhraní.

EZ-kit lite je navržen za účelem demonstrovat možnosti a schopnosti procesoru ADSP-21369. Blokové schéma kitu ilustruje obrázek 18.



Obrázek 18 - Architektura vývojového kitu [13]

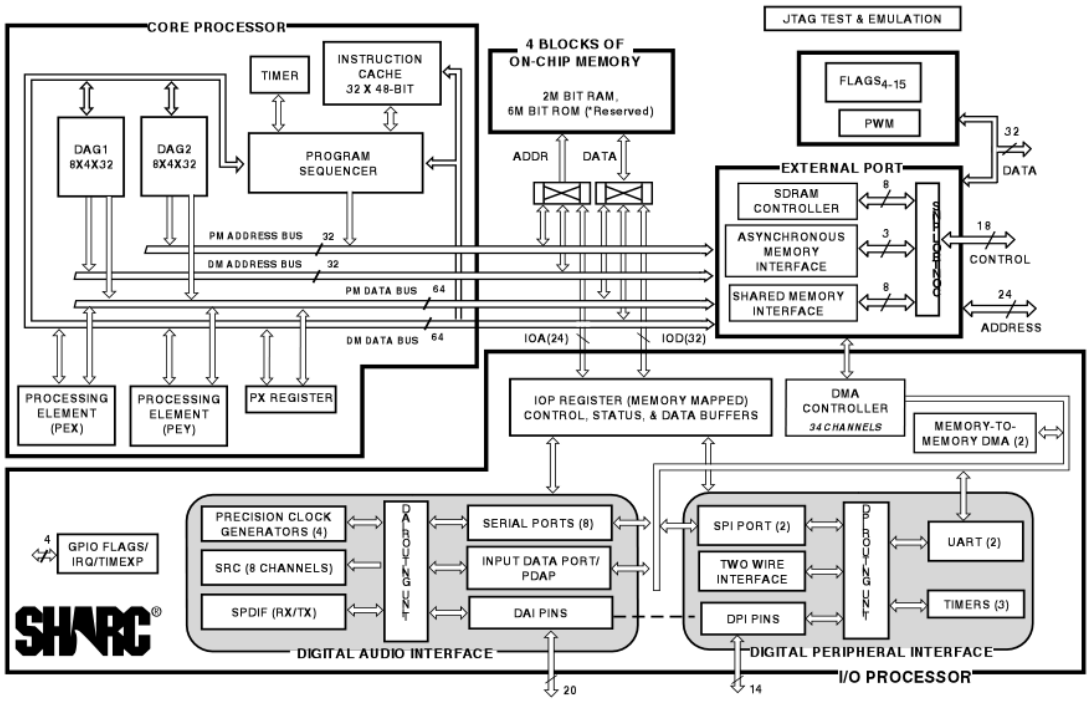
O napájení se stará spínaný regulátor napětí, který zajišťuje 1,3 V pro napájení jádra procesoru, 3,3 V pro napájení vstupně-výstupních bran procesoru a některých periferií a nakonec 5 V pro ostatní periferie.

Za zmínku stojí upozornit na 8 LED diod pro zobrazení stavu programu a 4 tlačítka pro uživatelskou interakci. Ladicí rozhraní umožňuje krokovat program a prohlížet, případně modifikovat, interní a externí paměť přímo na kitu. Pokud je připojen běžný emulátor JTAG (např. ADZS-USB-ICE), integrované USB ladicí rozhraní je zakázáno.

V dodávané dokumentaci je kompletní schéma celého zařízení (kromě USB ladicího rozhraní), takže při vývoji vlastní aplikace je možné z uvedených schémat vycházet. Najdeme zde i popis signálů rozšiřujícího rozhraní s 90 piny, jehož konektory jsou umístěny ze spodní strany vývojové desky.

5.1 ADSP-21369

Některé vlastnosti osazeného procesoru už byly zmíněny výše. Na obrázku 19 můžeme vidět blokové schéma procesoru ADSP-21369, které si následně stručně popíšeme.



Obrázek 19 - Blokové schéma ADSP-21369 (zdroj: [14])

V levé horní části obrázku 19, kde se nachází blok jádra (*core processor*), stojí za povšimnutí blok instrukční cache, která je právě jednou z výhod architektury SHARC. Její velikost 32 VLIW instrukcí o délce 48 bitů se může zdát malá v porovnání s cache, jakou mají procesory dnešních osobních počítačů. Musíme si však uvědomit, že instrukční cache se využije hlavně při výpočtu číslicových filtrů, nebo FFT, kde je samotný algoritmus velice jednoduchá smyčka opakujících se několika málo instrukcí.

V jádře můžeme také najít bloky PEX a PEY. Pod každým z nich se skrývá ALU, násobička, shifter a sada datových registrů. Existence těchto dvou nezávislých bloků v jádře umožňuje zpracovat stejným způsobem dvojí data – SIMD.

Procesor také obsahuje 2 dvouportové paměti, které umožňují souběžné čtení/zápis dat, případně instrukcí, z pamětí a zároveň paměťové DMA přenosy na pozadí. Nechybí ani rozhraní pro připojení externí paměti s řadičem SDRAM.

V bloku I/O procesoru se nachází speciální rozhraní DAI určené pro připojení audio periferií, jako jsou audio kodeky, A/D a D/A převodníky, nebo rozhraní S/PDIF. Nalezneme zde i další blok DPI, který slouží k připojení dvou SPI portů, dvou univerzálních asynchronních vysílačů/přijímačů, dvouvodičového vedení TWI, 12 flagů a třech časovačů pro obecné použití [14].

5.2 Audio kodek AD1835A

AD1835A je jednočipový kodek obsahující 4 stereo D/A převodníky a jeden stereo A/D převodník. Každý D/A převodník se skládá z digitálního interpolačního filtru, vícebitového sigma-delta modulátoru, který představuje patentovanou technologii firmy Analog

Devices, a z výstupní části se spojitým časem. Každý D/A převodník má nezávislé ovládání hlasitosti a funkci ztlumení bez nežádoucích klikavých zvuků. Oba D/A převodníky obsahují 24bitové konverzní kanály s vícebitovými sigma-delta modulátory a decimačními filtry.

Audio kodek disponuje flexibilním sériovým rozhraní pro jednoduché připojení k DSP čipům, AES/EBU přijímačům a konvertorům vzorkovací frekvence. Sériový port je možné nastavit do čtyř různých módů. Konfigurace kodeku se provádí přes SPI kompatibilní rozhraní.

6. Výběr způsobu implementace

6.1 Výběr programovacího jazyka

Při výběru vhodného programovacího jazyka pro vytvoření algoritmu byly na výběr tři možnosti. Jazyk symbolických adres, jazyk C a jazyk C++. Vzhledem k tomu, že dekódování formátu MP3 je značně komplikovaná záležitost vyžadující uchovávat složité struktury dat, jako jsou informace z hlavičky a hlavně dodatečné informace, je vhodnější použití vyššího programovacího jazyka. I přesto, že by v některých případech bylo vhodné použít OOP, byl nakonec zvolen pro svou jednoduchost jazyk C.

6.2 Volba vhodných datových typů

Jak už bylo zmíněno, při dekódování MPEG-1 layer 3 je vhodné uchovávat některé informace se složitou strukturou dat. Proto byly pro informace z hlavičky a dodatečnou informaci navrženy vhodné datové struktury. Jelikož se tyto datové struktury používají napříč celým algoritmem, byl jejich návrh velice důležitý. V případě datové struktury pro uložení dodatečné informace se jedná o několik vnořených struktur.

Zde je pro ilustraci datová struktura pro uchování dodatečné informace.

```
typedef struct {
        unsigned main_data_begin;
        unsigned private_bits;
        struct {
           unsigned scfsi[4];
                 struct gr_info_s {
                          unsigned part2_3_length;
                          unsigned part2_length;
                          unsigned big_values;
                          unsigned global_gain;
                          unsigned scalefac_compress;
                          unsigned window_switching_flag;
                          unsigned block_type;
                          unsigned mixed_block_flag;
                          unsigned table_select[3];
                          unsigned subblock_gain[3];
                          unsigned region0_count;
                          unsigned region1_count;
                          unsigned preflag;
                          unsigned scalefac_scale;
                          unsigned count1table_select;
                          } gr[2];
                 } ch[2];
        } side_info_t;
```

Jak je možné vidět na příkladu struktury side_info_t, jedná se o 3 vnořené datové struktury s obsahem, který doporučuje standard [1].

6.3 Volba aritmetiky

Při výběru aritmetiky pro výpočty bylo na výběr mezi aritmetikou s plovoucí, nebo pevnou řádovou čárkou. Jelikož má podle [14] použitý procesor ADSP-21369 stejný výkon v plovoucí i pevné řádové čárce, byla pro jednoduchost implementace zvolena právě plovoucí řádová čárka.

Veškeré výpočty byly prováděny ve formátu s plovoucí řádovou čárkou s jednoduchou přesností, jelikož je pro veškeré výpočty dostačující. Výpočty ve formátu float jsou na platformě SHARC podporovány hardwarem, takže poskytují mnohem vyšší výkon, než výpočty ve formátu double. Jako výstupní formát byl zvolen 16bitový PCM výstup (oproti dnes běžnému 24bitovému). Změna výstupu na 20, nebo 24bitovou přesnost vyžaduje jen velice malé změny ve zdrojovém kódu.

Dostupné dekodéry MP3 jsou většinou napsané v plovoucí řádové čárce. Najdou se však i takové, jako například MAD (MPEG Audio Decoder) [15], které jsou založeny plně na pevné řádové čárce a díky tomu jsou předurčené k nasazení na CPU bez aritmetiky s plovoucí řádovou čárkou. Nabízí se otázka. Proč nevyužít MAD pro implementaci na DSP? Dekodér MAD je určen pro PC a nasazení na DSP by vyžadovalo úpravy zdrojových kódů. Jelikož je kód dekodéru MAD po algoritmické stránce velice optimalizovaný, je velice nepřehledný a funkce programu je jen velice těžko pochopitelná. Aby bylo využito všech možností architektury SHARC, bylo by dále nutné přepsat časově kritické operace do jazyka symbolických adres. Z důvodu špatné čitelnosti zdrojového kódu by však tato operace mohla být velice komplikovaná. MAD také pro úsporu výpočetního výkonu používá mnoho tabulek s předem vypočtenými hodnotami. Velikost těchto tabulek je v řádu stovek kilobitů, což je podstatná část vnitřní paměti použitého procesoru. Z výše uvedených důvodů byla zvolena možnost napsání vlastního MP3 dekodéru kompletně od začátku.

7. Popis navržené rutiny

Celý program se skládá z funkcí, které více, či méně, kopírují funkce bloků z obrázku 1. Před spuštěním programu je nutné přes JTAG rozhraní nahrát do SDRAM paměti na desce kitu zdrojový soubor na adresu odpovídající proměnné *file*. Dekódovací algoritmus je založen na smyčce, která provádí všechny potřebné dekódovací operace, dokud níže popsaná funkce *find_frame* najde v paměti další platnou hlavičku MP3 souboru.

Veškeré zdrojové soubory jsou uloženy na přiloženém CD.

7.1 Omezení

Navržená rutina je schopna korektně dekódovat pouze soubory MPEG-1 layer 3 s konstantní přenosovou rychlostí (bitrate) vyšší, než 192 kb/s a vzorkovací frekvencí 48 kHz, ať už mono, či stereo (vyjma joint stereo). Rovněž není prováděná kontrola chráněných bitů v případě přítomnosti ochranného CRC kódu.

7.2 Synchronizace a kontrola chyb

O synchronizaci, čili nalezení synchronizační sekvence, se stará funkce *find_frame* umístěná v souboru frame.c. Funkce má 2 parametry, prvním z nich je pointer na uložený soubor v SDRAM paměti vývojového kitu a druhým parametrem je adresa další hlavičky. Při prvním průchodu je tato adresa nulová, ale po dekódování hlavičky známe délku fyzického rámce a víme, kde začíná další fyzický rámec. Funkce *find_frame* tedy dostane jako parametr předpokládanou adresu dalšího fyzického rámce a provádí hledání až od této adresy. Pokud funkce *find_frame* nenalezne žádnou další platnou hlavičku a narazí na konec souboru, vrátí hodnotu *false* a výše popsaná smyčka se ukončí. Následně program končí v nekonečné smyčce.

Navržený program kontrolu chyb pomocí CRC neprovádí a případné ochranné bity ignoruje.

7.3 Dekódování hlavičky

Po nalezení začátku fyzického rámce funkcí *find_frame*, jsou do bufferu uloženy 4 bajty hlavičky, a můžeme přistoupit k dekódování.

Jelikož jsou informace uložené v MP3 datovém proudu výhradně bitově orientované, byla vytvořena funce *get_bits*, která umožňuje postupně číst jednotlivé bity z bufferu bez složitého maskování bitů, které je často zdrojem chyb. Funkce *get_bits* je popsána v kapitole 6.4.

Dekódování hlavičky zajistí funkce *decode_header* z frame.c, která má dva parametry. Prvním parametrem je pointer na strukturu typu *header*, do níž se ukládají dekódovaná data z hlavičky a druhým parametrem je adresa hlavičky, která se rovněž uloží do datové struktury pro pozdější využití.

```
typedef struct {
   int crc;             // 1
   int bitrate;         // 4
   int sampl_freq;      // 2
   int padding;         // 1
   int priv;            // 1
   int mode;            // 2
   int mode_ext;        // 2
   int copyright;       // 1
   int original;        // 1
   int emphasis;        // 2
   int addr;            // adresa hlavičky/fyz. rámce
   int frame_len;       // délka fyz. rámce
   int si_len;          // délka sideinfo
   int sfreqID;         // ID sampl frekvence
} header;
```

Před dekódováním hlavičky se nejdříve inicializuje ukazatel pro funkci *get_bits*, viz níže. Následně můžeme přistoupit k vyčítání bitů z bufferu. Prvních 15 bitů nemá pro nás informační hodnotu, protože se jedná o 11 bitů synchronizační sekvence, 2 bity pro definici verze MPEG (předpokládáme verzi MPEG-1) a zbylé 2 bity jsou na definici vrstvy (předpokládáme layer 3).

Následně čteme postupně bity pro získání informací do struktury *header*. Potřebný počet bitů pro jednotlivá pole hlavičky je uveden v komentáři definice struktury.

7.4 Pomocná funkce get_bits na čtení bitů bez maskování

Díky této funkci je dekódování MP3 formátu mnohem jednoduší, podobné řešení v jazyce symbolických adres by bylo mnohokrát komplikovanější.

Všechny potřebné funkce pro výčet bitů z bufferu se nacházejí v souboru bitstream.c. Buffer, ze kterého funkce *get_bits* čte, je pole datových typů *char*, takže délka jednotlivé položky pole je 8 bitů. Z toho důvodu nám stačí jen 8 různých hodnot pro maskování.

Pro orientaci v bufferu se používá proměnná *pos*, která ukazuje na bit, od kterého dál budeme číst počet bitů uvedených v parametru funkce *get_bits*. Následuje popis jednotlivých funkcí.

bs_init

Funkce *bs_init* nemá žádné parametry, ani návratové hodnoty. Slouží pouze k inicializaci výše zmíněného ukazatele, tzn., nastaví proměnnou *pos* na hodnotu 0.

bs_get_pos

Tato funkce bez parametru vrací aktuální pozici ukazatele bufferu v bitech.

bs_start_count

Funkce *bs_start_count* slouží k uložení aktuální pozice ukazatele do další proměnné *citac* a později umožní určit počet vyčtených bitů od okamžiku volání této funkce. Tato vlastnost se hodí například pro zjištění délky části s měřítky (*part2_length*), která má proměnnou délku.

bs_get_counter

Tato funkce bez parametru vrací počet bitů vyčtených od okamžiku volání funkce *bs_start_count*.

bs_shift_ptr

Pomocí funkce *bs_shift_ptr* posuneme ukazatel o počet bitů udaných v parametru. Záporná hodnota značí posun zpět, kladná posun vpřed.

get_bits

Nakonec nejdůležitější funkce *get_bits* s parametrem udávajícím počet bitů k vyčtení. Z aktuální pozice ukazatele se zjistí, který bajt z bufferu obsahuje žádané bity. Následně se kombinací posunů, maskování a sčítání získá výsledek. Algoritmus si samozřejmě poradí i s případem, kdy je požadovaný počet bitů rozdělen do dvou a více bajtů.

Tato funkce se využívá pro kompletní dekódování komprimovaných spektrálních složek z MP3 bitového proudu. Bez ní by byl celý algoritmus dekódování formátu MP3 složitější, méně přehledný a také více náchylný k chybám kvůli neustálému maskování bitů.

7.5 Získání dodatečné informace

Před dekódováním dodatečné informace potřebujeme vědět, kolik zvukových kanálů obsahuje náš MP3 soubor. Jak bylo zmíněno v části 2.4, délka dodatečné informace je závislá právě na počtu kanálů. O proces získání dodatečné informace se stará funkce *decode_sideinfo (*umístěná v souboru frame.c) se třemi vstupními parametry. Prvním z nich je ukazatel na MP3 soubor v paměti, druhým je ukazatel na strukturu informací z hlavičky a třetím je ukazatel na strukturu dodatečné informace.

Před samotným dekódováním dodatečné informace se musí nejdříve získat potřebný počet bajtů z paměti do bufferu. Na základě informací z hlavičky se určí potřebná délka zakódované informace a počet bajtů od začátku hlavičky, které je třeba přeskočit. V případě jednokanálového módu je délka dodatečné informace 17 bajtů, v ostatních případech je to 32 bajtů. Vždy musíme přeskočit 4 bajty hlavičky, a pokud je přítomen ochranný kód (ochranný bit má hodnotu 0), musíme navíc přeskočit další 2 bajty. Jakmile známe pozici a délku zakódované dodatečné informace, můžeme ji přesunout z SDRAM do bufferu, kde je připravena k dekódování.

Následně už jen čteme požadovaný počet bitů pomocí už mnohokrát zmíněné funkce *get_bits*. Nejprve přečteme *main_data_begin* a následně *private_bits*. Poté následují čtyři hodnoty *scfsi* pro každý kanál. Nakonec čteme postupně data pro obě granule, v každé z nich jsou informace pro jednotlivé kanály.

Při čtení dodatečné informace dochází k testování mnoha podmínek, zejména hodnoty *windows_switching_flag*. Tento flag ovlivňuje skladbu dodatečné informace, mimo jiné i počet dekódovacích Huffmanových tabulek.

7.6 Získání hlavních dat

Získání hlavních dat, která obsahují zakódované vzorky spektrálních složek, není vzhledem k negativnímu ofsetu (*main_data_begin*) úplně jednoduchou záležitostí. Jak již bylo zmíněno, data logického rámce mohou být díky hodnotě *main_data_begin* umístěna až o několik fyzických rámců zpět. Při jejich získávání musíme zároveň přeskočit všechny hlavičky, dodatečné informace a případně i ochranná slova, která nám stojí v cestě. Zároveň musíme znát hodnotu *main_data_begin* z následujícího fyzického rámce, abychom mohli určit, kde aktuální logický rámec končí.

Toto vše řeší funkce *get_main_data,* ze souboru frame.c, s třemi vstupními parametry. Ukazatel na soubor MP3 v paměti, ukazatel na informace z hlavičky pro zjištění počtu kanálů a tím délky dodatečné informace, kterou je třeba přeskočit, a nakonec ukazatel na strukturu s dodatečnou informací, ve které je uložena hodnota *main_data_begin*.

Získávání hlavních dat začíná zjištěním adresy jejich začátku. Toto se provede odečtením hodnoty *main_data_begin* od počáteční adresy aktuální hlavičky. Následně se bajt po bajtu kopíruje do bufferu s tím, že pokud se narazí na začátek hlavičky, přeskočí se potřebný počet bajtů, které do hlavních dat nepatří.

Po zkopírování hlavních dat do bufferu může následovat jejich dekódovaní. Jak víme z obrázku 1, hlavní data obsahují vždy dvě granule a v každé z nich mohou být informace až o dvou kanálech.

Jelikož soubor MP3 obsahuje synchronizační sekvenci (11 bitů s log. 1) často i v hlavních datech, není možné v případě přetečení logického rámce do více předchozích fyzických rámců správně určit a přeskočit hlavičky a dodatečné informace. Z tohoto důvodu navržený algoritmus dekóduje správně pouze MP3 s bitrate 192 kb/s a více, protože tento bitrate zajišťuje podle vztahu 3.1, že délka fyzického rámce je vyšší, než maximální hodnota *main_data_begin* (511 bajtů). Díky tomu se při čtení hlavních dat přeskakuje vždy maximálně jen jedna hlavička a to pouze hlavička aktuálního rámce, jehož polohu známe. V případě vylepšování algoritmu by bylo tuto funkci doplnit a metodu správné detekce hlaviček na jiném principu, který by mohl podporovat i variabilní bitrate.

7.7 Získání měřítek

Měřítka jsou umístěna vždy na začátku každé granule (v závislosti na *scfsi* nemusí druhá granule některá měřítka obsahovat). Měřítka mají proměnnou délku, která se řídí výpočty podle vztahů z 3.6. Způsob jak měřítka z hlavních dat přečíst je rozdílný pro dlouhé bloky, krátké bloky i pro smíšené bloky. O jaký blok se jedná lze zjistit z dodatečné informace. Při čtení měřítek se uplatní parametr *scalefac_compress*, který udává délku měřítek *slen1* a *slen2*, a také již zmiňovaný parametr *scfsi*.

O získání měřítek se stará funkce *get_scale_factors* (z layer3.c) se čtyřmi vstupními parametry. Prvním vstupním parametrem je datová struktura měřítek (*scalefac_t*), která je navržena k ukládání měřítek pro dlouhé (pole l), krátké (pole s), i složené bloky (kombinace obou polí). Druhým parametrem je dodatečná informace a za nimi následuje index aktuálního kanálu a index aktuální granule.

```
typedef struct {
        int l[23];          // [subpásmo]
        int s[3][13];       // [okno][subpásmo]
        } scalefac_t[2];  // [kanál]
```

Po získání měřítek se, pomocí již zmíněné funkce *bs_get_counter,* zjistí délka části měřítek (*part2_length*). Tato hodnota se použije pro výpočet délky oblasti s bity zakódovanými Huffmanovým kódem (*part3_length*) jako rozdíl hodnoty *part2_3_length* z okrajové informace a právě zjištěné délky oblasti měřítek.

7.8 Dekódování Huffmanových bitů

Po extrahování souboru měřítek z počátku granule můžeme přistoupit k dekódování spektrálních složek. Toto má za úkol funkce *huffdec* (z layer3.c) s dvěma vstupními parametry. Tím prvním je část dodatečné informace pro právě zpracovávanou granuli a druhým parametrem je ukazatel na pole dekódovaných hodnot. Tyto hodnoty se podle standardu nazývají *is*.

Huffmanův dekodér překládá kód s proměnnou délkou na spektrální čáry. Dekodér používá tabulky definované ve standardu [1],[7], kde je uvedeno jak přečtené bity překládat na hodnoty amplitud frekvenčního spektra. Při návrhu algoritmu Huffmanova dekodéru byly na výběr tři možnosti.

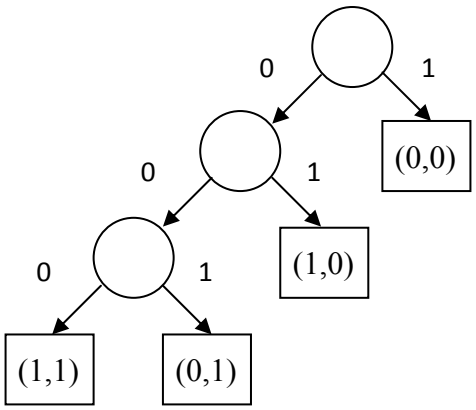
7.8.1 Přímé vyhledávání v tabulce

Metoda přímého vyhledávání vyžaduje obrovské tabulky. Počet hodnot každé tabulky je 2^b, kde *b* je délka nejdelšího kódu tabulky. Nejdelší kód v Huffmanově tabulce č. 24 je 11 bitů. To znamená, že vytvořená tabulka pro tuto metodu by měla velikost 2^{11}, tzn. 2048 záznamů, i když ve skutečnosti obsahuje pouze 256 různých hodnot. Tuto metodu lze z důvodu vysoké redundance použít pouze při dostatku paměti a nedostatečném výkonu procesoru.

Při dekódování je přečteno právě *b* bitů a zjištěná hodnota je použita jako přímý index tabulky. V případě, že Huffmanův kód má menší délku než *b*, využijí se přebytečné bity pro dekódování další hodnoty.

7.8.2 Prohledávání binárního stromu

Tabulky pro dekódování Huffmanova kódu mohou být převedeny do podoby binárních stromů. Každý strom potom představuje jednu tabulku. Stromy se procházejí v souladu s přečtenými bity, kdy ‚0' znamená jít vlevo a ‚1' znamená jít vpravo. Celé kódové slovo je nalezeno, jakmile se narazí na list stromu (označen obdélníkem). Listy stromu obsahují dekódované hodnoty. Huffmanova tabulka transformovaná do podoby binárního stromu je vyobrazena na obrázku 20.



Obrázek 20 - Huffmanova tabulka č. 1 v podobě binárního stromu

7.8.3 Kombinované dekódování

Tato metoda, popsaná v [8], je na implementaci poměrně jednoduchá. Problémem může být však generování složitých tabulek. Jedná se vlastně o kombinaci obou předchozích metod s tím, že je kompromisem, mezi paměťovou náročností a složitostí algoritmu.

Jedná se o velice složité LUT, jejichž hodnotami jsou datové struktury. Hodnotou mohou být buď přímo datová struktura obsahující velikosti dekódovaných spektrálních složek, nebo struktura s informacemi pro další krok dekódování.

Celý algoritmus funguje následovně. Zvolíme tabulku v závislosti na parametru *table_select,* z dodatečné informace, a aktuálním regionu. Tabulka není jednoduché pole, ale datová struktura, takže po výběru tabulky známe velikost *linbits* a počet bitů, které je nejprve nutné přečíst z hlavních dat (říkejme této hodnotě *startbits*). Nejprve přečteme z hlavních dat právě tolik bitů, jako je hodnota *startbits*. Získanou informaci použijeme jako přímý index speciální LUT. Nyní mohou nastat dva případy. Datová struktura na právě přečteném indexu má nastavenou hodnotu *final* na *‚1'*, to znamená, že jsme právě dekódovali hodnoty *x* a *y* (případně ještě *v* a *w*) a můžeme si jejich hodnoty přečíst z navrácené datové struktury. V opačném případě, kdy je hodnota *final* rovna nule, dekódování neskončilo a musíme v něm pokračovat. Vrácená struktura nám dá, mimo jiné, informace *bits* a *offset*. Hodnota *bits* určuje, kolik dalších bitů z hlavních dat musíme přečíst v dalším

kroku. *Offset* je číslo, které je nutné přičíst k novému indexu, který získáme přečtením dalších *bits* bitů. Tento krok opakujeme do té doby, než je v navrácené struktuře nastaven flag *final* na *‚1‘*.

Pro srovnání, délka největší Huffmanovy tabulky pro tuto metodu je jen 386 hodnot oproti 2048 v metodě popsané v 7.8.2. Následuje ukázka složité datové struktury pro tabulky z oblasti *big_values*.

```
union huffpair {
  struct {
    unsigned short final  :  1;
    unsigned short bits   :  3;
    unsigned short offset : 12;
  } ptr;
  struct {
    unsigned short final  :  1;
    unsigned short hlen   :  3;
    unsigned short x      :  4;
    unsigned short y      :  4;
  } value;
  unsigned short final    :  1;
};
```

Více ve zdrojových souborech huffman.h a huffman.c.

7.8.4 Zvolená metoda

Pro svou relativní jednoduchost byla v navrženém algoritmu dekódování MP3 zvolena metoda kombinovaného dekódování. Vlastní generování tabulek je však složité. Proto jsou v navrženém algoritmu použity soubory huffman.c a huffman.h, které jsou součástí dekodéru MAD zmíněného výše. Dekodér MAD je uvolněn pod licencí GNU/GPL a využívá jej například i projekt DSPdap [16] založený na 16bitovém celočíselném procesoru Texas Instruments.

Při získávání vzorků spektra pomocí Huffmanovy tabulky algoritmus postupuje podle popisu z části 7.8.3.

7.9 Rekvantizace

Proces rekvantizace musí být proveden na každém dekódovaném vzorku. Zde se uplatní měřítka získaná z hlavních dat a také několik hodnot z dodatečné informace, jako je *global_gain*, *preflag*, *scalefac_scale* a případně i *subblock_gain*, více o těchto hodnotách v části 3.4.1.

Jak je uvedeno v 3.8, rekvantizace jednoho vzorku vyžaduje výpočet tří různých mocnin. Získání těchto tří výsledků se dá docílit více metodami, které se liší paměťovou a výpočetní náročností.

Připomeňme si vztahy pro rekvantizaci vzorků.

Krátké bloky:

$$xr[i] = sign(is[i]) * |is[i]|^{\frac{4}{3}} * 2^A * 2^B, \tag{3.26}$$

kde pro A a B platí:

$$A = \frac{1}{4} * (global\_gain[gr] - 210 - 8 * subblock\_gain[okno][gr]), \tag{3.27}$$

$$B = -(scalefac\_scale * scalefac\_s[gr][ch][sfb][okno]. \tag{3.28}$$

Dlouhé bloky:

$$xr[i] = sign(is[i]) * |is[i]|^{\frac{4}{3}} * 2^A * 2^B, \tag{3.29}$$

kde pro A a B platí:

$$A = \frac{1}{4} * (global\_gain[gr] - 210), \tag{3.30}$$

$$B = -scalefac\_scale * (scalefac\_l[sfb][ch][gr][okno] + preflag[gr] * pretab[sfb]). \tag{3.31}$$

7.9.1 Přímý výpočet

Přímý výpočet pomocí funkce *pow* z knihovny math.h je nejjednodušší možností. Na druhou stranu je tato možnost nejvíce výpočetně náročná. Jelikož jsou při rekvantizaci potřebné hned tři výpočty mocniny, je dobré se poohlédnout po dalších možnostech. Při použití celočíselných DSP tato možnost nepřipadá v úvahu a je nutné najít další řešení.

7.9.2 Hodnota získaná z tabulky

Při úvahách o vytvoření LUT pro tuto operaci je dobré zjistit maximální počet hodnot, kterých může daný výsledek nabývat. V případě výpočtu $is^{4/3}$ zjistíme maximální počet možných výsledků snadno. Jak už jsme se zmínili, hodnota *is*, čili hodnota získaná dekódováním Huffmanových bitů, může nabývat hodnot -8206 až 8206. Díky vlastnostem mocniny můžeme vytvořit tabulku pouze pro 8207 hodnot. Taková tabulka s výsledky ve formátu float s jednoduchou přesností (nebo ve zlomkovém formátu 1.31) má potom přibližně 256 kb. Jestliže je k dispozici dostatek paměti, je vhodné vytvořit tabulku i pro záporné hodnoty. Celý proces rekvantizace se tím urychlí. Pokud nemáme dostatek paměti, ani rychlý procesor, můžeme využít aproximační metody popsané dále.

V případě výpočtu hodnot $2^{(0,25*A)}*2^{-B}$ nedostaneme více než 384 hodnot [8]. V tomto případě se tedy hodí použití tabulky i pro systémy s omezeným paměťovým prostorem. Velikost výsledné LUT můžeme ještě zmenšit (až na 196) zaokrouhlením velice malých hodnot ($< 2^{-35}$), aniž bychom ztratili přesnost výsledku.

7.9.3 Newtonova iterační metoda

Vztah $y = x^{4/3}$ můžeme přepsat do podoby $y^3 = x^4$. Tento tvar je vhodný pro Newtonovu metodu hledání kořenů, která nám poskytne přibližnou hodnotu $x^{4/3}$.

Výsledek je počítán pomocí postupných iterací a tím se snižuje velikost reziduální chyby $|y- x^{4/3}|$. Iterační vztah je následující:

$$y_{n+1} = y_n - \frac{y_n^3 - x^4}{3y_n^2} = \frac{2y_n^3 + x^4}{3y_n^2}. \tag{7.1}$$

Vztah je přepsán do konečného tvaru, který minimalizuje chyby vzniklé při výpočtech v plovoucí řádové čárce.

Počáteční hodnota *y* pro *n*=0 iteračního vztahu velice závisí na počtu iterací, který je nutný k dosažení požadované přesnosti. V našem případě je dostatečná přesnost vyšší, než 16 bitů. Vhodná počáteční hodnota y_0 je vypočtena pomocí polynomické regresní funkce druhého řádu:

$$y_0 = a_0 + a_1 * x + a_2 * x^2. \tag{7.2}$$

Tento prvotní odhad zajišťuje určení počáteční hodnoty $x^{4/3}$ s nejvyšší možnou přesností pro $0 < x < 8207$. Takový prvotní odhad hodnoty zajišťuje, že požadované přesnosti výpočtu mocniny je dosaženo již při třetí iteraci.

7.9.4 Zvolená metoda

S ohledem na poměrně vysoký výkon použitého DSP byla v algoritmu zvolena metoda přímého výpočtu. O rekvantizaci se stará funkce *dequantize_sample* (z layer3.c) s pěti vstupními parametry. Prvním z nich je ukazatel na pole vstupních hodnot, druhým je pole výstupním rekvantizovaných hodnot. Třetí parametr je ukazatel na strukturu dodatečné informace aktuální granule, čtvrtým parametrem je struktura dat hlavičky a poslední parametr je index aktuálního kanálu.

Při samotné rekvantizaci je nutné hlídat hranice měřítkových subpásem uvedených na obrázku 8. S každou změnou subpásma nastupuje totiž rozdílná hodnota měřítka. Hranice subpásem se liší podle vzorkovací frekvence a podle typu bloku. Délky a umístění jednotlivých subpásem měřítek jsou uvedeny v [1] a [7]. Výstupní hodnoty jsou navíc řazeny nejdříve podle subpásma (nejedná se o subpásmo měřítek s proměnnou délkou, ale o sub-

pásmo po 18 vzorcích) a potom podle vzorku. Tím dostaneme výstupní pole 32 subpásem, každé s 18 vzorky.

Funkce *dequantize_sample* byla převzata z referenčních zdrojových kódů, které poskytuje Frauhofer Institut – autor kódovací techniky MPEG-1 layer 3. Jak můžeme vidět ve vztazích 3.7 a 3.10, nabízí se velice jednoduchá metoda optimalizace výpočtu. V případě krátkých oken je tento výraz stejný pro celou granuli, tzn. pro všech 576 vzorků, takže stačí provést výpočet této mocniny pouze jednou. Tím se ušetří 575 výpočtů mocniny. V případě krátkých oken je potřeba vypočítat 3 různé hodnoty pro každé okno, ale přesto se ušetří výpočet 573 mocnin. Podle testů se tím algoritmus zrychlil cca o 1,5%.

7.10 Přerovnání spektra

Tento krok je velice jednoduchý. Důvod přerovnání spektra je vysvětlen v kapitole 3.9. O přerovnání spektra se stará funkce *reorder* (z layer3.c), do které vstupují rekvantizované vzorky, dodatečná informace aktuální granule a informace z hlavičky.

Samotné přerovnání má 3 varianty. První z nich je přerovnání složeného bloku. Tato operace je nejsložitější. Jak již bylo zmíněno, ve složených blocích jsou nejprve 2 subpásma zakódovaná jako dlouhý blok a zbylá subpásma jako krátký blok. Z toho důvodu počáteční 2 subpásma nepřerovnáme. Další možností je čistě krátký blok. V něm se přerovnání provádí podle popisu z 3.9. Poslední možností je dlouhý blok. V něm se žádné vůbec nepřerovnávají.

Tato funkce byla rovněž převzata z referenčních kódů Frauhofer Institutu a nebyla již dál nijak upravována, protože zde nejsou žádné možnosti optimalizace.

7.11 Zpracování sterea

Nyní by mělo následovat zpracování joint sterea popsané v části 3.10. Tato rutina má však některá omezení a dekódování joint stereo MP3 nepodporuje. Klasické stereo nevyžaduje žádné další kroky, protože je plná informace pro pravý a levý kanál uložená v příslušné granuli.

7.12 Redukce aliasů

Z dříve uvedených důvodů, je třeba před syntézou u dlouhých bloků redukovat aliasing vzniklý v analyzující bance polyfázových filtrů. Redukci aliasů zajišťuje funkce *antialias* (z layer3.c) s třemi vstupními parametry. Prvním z nich je vstupní pole přerovnaných vzorků, druhým je výstupní pole vzorků a poslední je dodatečná informace aktuální granule.

Funkce si nejprve vytvoří dvě malé LUT s hodnotami *cs* a *ca* (viz 3.11). Následné je zkontrolován typ bloku, v případě krátkého bloku se redukce aliasů neprovádí. V ostatních dvou případech je proveden výpočet osmi motýlků mezi všemi sousedními subpásmy

podle obrázku 10. Tato funkce byla opět převzata z referenčního zdrojového kódu a byla použita bez dalších úprav.

7.13 Hybridní syntéza

Pod pojmem hybridní syntéza se ukrývá kombinace IMDCT a následné překrývání vzorků. Proces překrývání vzorků je jen obyčejné sčítání, takže se budeme dále věnovat jen problematice IMDCT.

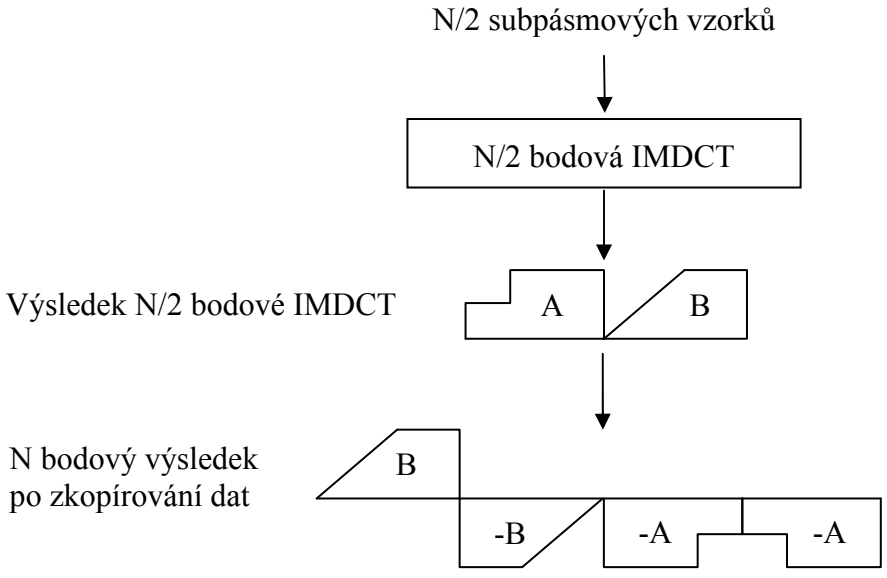
7.14 IMDCT

Funkce *inv_mdct*, umístěná v layer3.c, zajišťuje jak převod frekvenčních vzorků do časové oblasti, tak i následnou aplikaci váhovacího okna podle typu transformovaného bloku (viz obrázek 12). Funkce má 3 parametry, pole vstupních frekvenčních vzorků, pole výstupních časových vzorků a index čísla okna pro aplikaci váhovací funkce. Funkce navíc řeší ještě překrývání v případě krátkých bloků, kdy je třeba aplikovat tři inverzní transformace na 12 vzorků.

Pro zajištění menší výpočetní náročnosti je před výpočtem vytvořena LUT s hodnotami funkce kosinus a tím se IMDCT změní na pouhý součin hodnoty z LUT a velikosti spektrální složky a následný součet. I přesto se jedná o vysoký počet MAC operací. Použitý DSP nemám s tolika MAC operacemi problém, proto byla zvolena metoda přímého výpočtu. Pro zajištění menší výpočetní náročnosti se dá použít algoritmu rychlé IMDCT.

7.14.1 Rychlá IMDCT

Konstantinides v [17] dokázal, že při syntéze časových vzorků můžeme využít určité symetrie IMDCT a tím zjednodušit výpočet. Marovich navíc v [18] potvrdil, že toto zjednodušení lze uplatnit i při hybridní syntéze na dvanácti a 32bodovou IMDCT.

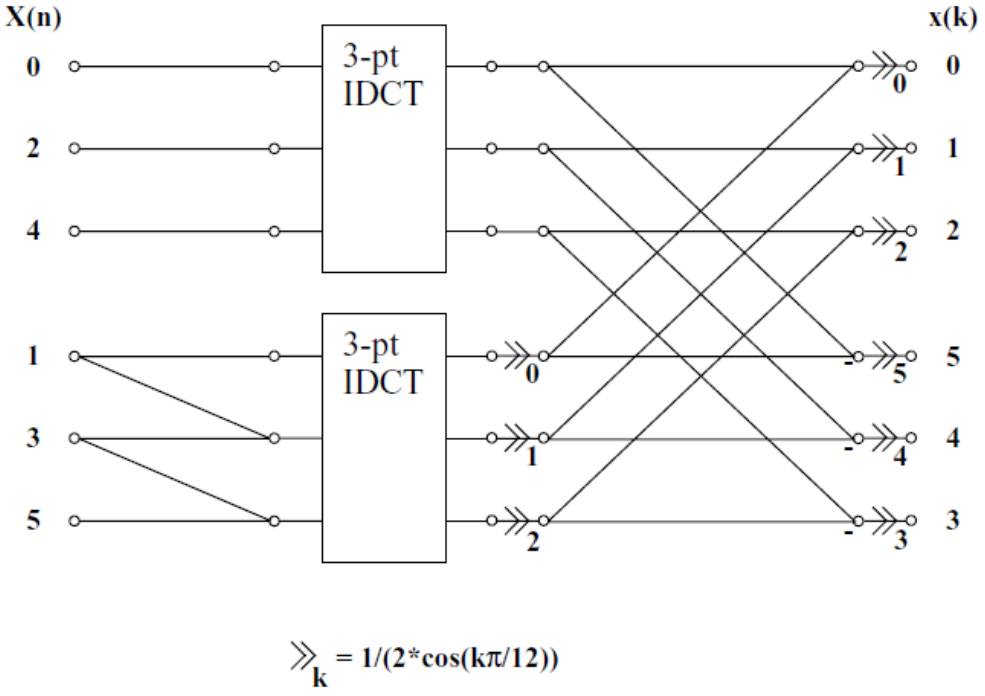


Obrázek 21 - Symetrie při výpočtu IMDCT (zdroj: [8])

Obrázek 21 zobrazuje nalezenou symetrii při výpočtu IMDCT. Takto získaný N bodový výsledek je identický s výsledkem IMDCT operace popsané ve standardu [1].

Díky nalezené symetrii nám stačí pouze optimalizovaný algoritmus výpočtu šesti, respektive 18 bodů IMDCT. Tyto body mohou být vypočítání metodou rozložení šesti a 18bodové IMDCT na tří, čtyř a pětibodová jádra IMDCT. Tuto metodu rozebírá Lee v [19].

Šestibodová transformace pro krátké bloky je rozložena na 2 tříbodové transformace, které mohou být vypočteny přímo. Více na obrázku 21. Podobně 18bodová transformace je rozložena na 2 části po devíti bodech, které jsou dále rozděleny na čtyři a pětibodové části a následně vypočteny.



$$\gg_k = 1/(2*\cos(k\pi/12))$$

Obrázek 22 - Rozložení 6bodové IMDCT na dvě 3bodová jádra (zdroj: [8])

7.14.2 Zvolená metoda

Funkce pro výpočet IMDCT byla rovněž převzata z referenčních kódů, ale byla dále optimalizována pomocí použití předem vypočítaných kosinových koeficientů i pro krátká okna a tím bylo docíleno zrychlení algoritmu o cca 6,6 %.

7.15 Banka polyfázových filtrů

7.15.1 Definice

Banka polyfázových filtrů konvertuje časové vzorky z hybridní syntézy (IMDCT) na výstupní vzorky PCM signálu. Konverze probíhá v následujících krocích:

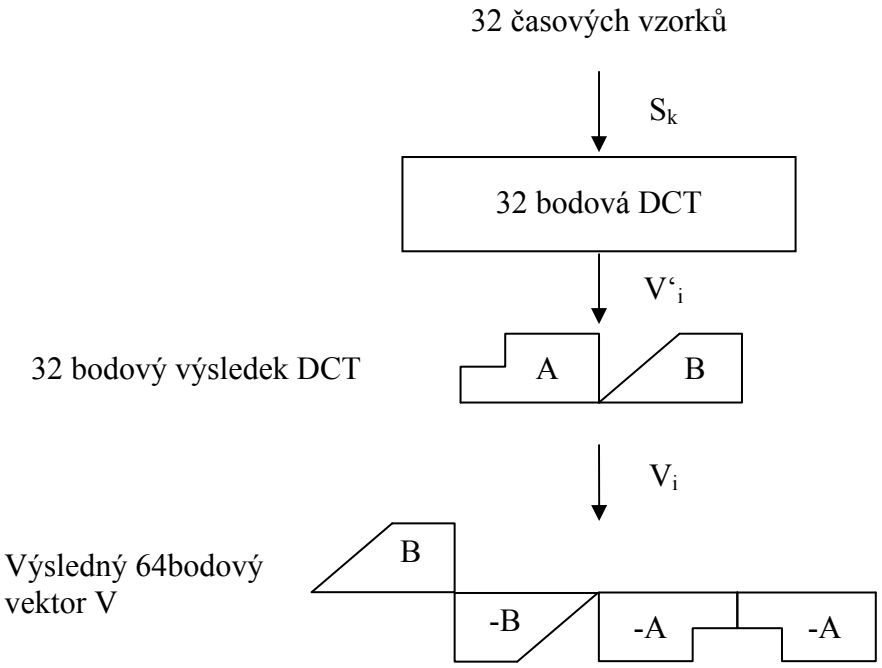
1. Konverze 32 vzorků ze subpásem do 64 hodnot *V* vektoru,
2. váhování hodnot z FIFO fronty *V* vektoru váhovací funkcí *D* a následné vytvoření vektoru *W*,
3. sečtení prvků vektoru *W* k vytvoření 32 výstupních PCM vzorků.

Realizace kroků 2 a 3 je přímočará a v literatuře se nenacházejí žádné možnosti optimalizace těchto procesů.

Konverze vstupních vzorků na 64bodový vektor *V* však už možnost optimalizace nabízí. Na výkonných platformách můžeme zvolit hodnotu přímého výpočtu, protože tato operace vyžaduje pouze dvě vnořené smyčky a výpočet hodnoty funkce kosinus.

7.15.2 Implementace rychlé 32bodové IMDCT

Konstantinides v [17] ukázal, že je při vytváření 64prvkového vektoru *V* možné použít výpočet IMDCT s polovičním počtem bodů a následně zkopírovat výsledek podle určeného vzorce k dosažení výsledku.



Obrázek 23 – Symetrie DCT při „matrixing“ operaci (zdroj: [8])

Díky nalezené symetrii se problém zužuje na nalezení rychlé implementace 32bodové DCT:

$$V_i' = \sum_{k=0}^{31} S_k * \cos\left[\frac{\pi}{64}(2k+1) * i\right] \quad \text{pro } i \in \langle 0; 31\rangle \tag{7.3}$$

Jeden z nejběžnějších algoritmů výpočtu rychlé DCT pro 2^m bodů popsal Lee v [19]. Jedná se o jednoduchou rekurzivní strukturu, kde je transformace rozložena na sudé a liché části:

$$X(n) = \sum_{k=0}^{N-1} x(k) * \cos\left(\pi(2k+1)\frac{n}{2N}\right) \quad \text{pro } n \in \langle 0; N-1\rangle \tag{7.4}$$

$$g(k) = x(k) + x(N-1-k) \tag{7.5}$$

$$h(k) = \frac{1}{2\cos\left(\pi(2k+1)\frac{n}{2N}\right)}(x(k) - x(N-1-k)) \tag{7.6}$$

$$G(n) = \sum_{k=0}^{\frac{N}{2}-1} g(k)\cos\left(\pi(2k+1)\frac{n}{N}\right), \qquad \text{pro } n \in \langle 0; N/2-1\rangle \tag{7.7}$$

$$H(n) = \sum_{k=0}^{\frac{N}{2}-1} h(k)\cos\left(\pi(2k+1)\frac{n}{N}\right), \qquad \text{pro } n \in \langle 0; N/2-1\rangle \tag{7.8}$$

Pro $n \in \langle 0; N/2-1\rangle$ platí:

$$X(2n) = G(n), \tag{7.9}$$

$$X(2n+1) = H(n) + H(n+1), \tag{7.10}$$

při splnění okrajové podmínky: $H\left(\frac{N}{2}\right) = 0$.

Liché a sudé části jsou dále děleny stejným způsobem, než dostaneme tak malé části, že mohou být přímo vypočteny, tzn., když N=2.

7.15.3 Zvolená metoda

V navrženém algoritmu byla při výpočtu syntezujícího filtru zvolena metoda přímého výpočtu DCT, která byla převzata z referenčních zdrojových kódů a dále optimali-

zována. Použitý DSP poskytuje dostatek výkonu pro tuto operaci. Běžný dekodér MP3 formátu stráví nejvíce času právě výpočtem syntezujícího filtru. V případě potřeby optimalizace algoritmu, je tedy nejlepší zaměřit se právě hlavně na tuto část, respektive na DCT operace. Aby se výpočet i bez složité optimalizace zrychlil, bylo při „matrixing“ operacích využito symetrie DCT uvedené v 7.15.2, takže se místo 64bodové DCT počítá pouze 32bodová DCT. Tato optimalizace vedla k rapidnímu zkrácení času potřebného k dekódování jedné granule. Více o optimalizaci je uvedeno v kapitole 8.

Syntéza časových vzorků z hybridního filtru do podoby výstupního PCM signálu je v navrženém algoritmu úkolem funkce s*ubband_synthesis* ze souboru layer3.c. Parametry této funkce jsou vstupní časové vzorky (vždy jeden z každého subpásma, je jich tedy 32), dále index kanálu a nakonec ukazatel pro uložení výstupních PCM vzorků.

Nejprve jsou provedeny alokace paměti pro potřebné proměnné. Následně je při prvním průchodu naplněna LUT s hodnotami funkce kosinus, pro urychlení dalších výpočtů. Dalším krokem je posun ukazatele FIFO fronty a připravení místa pro 64 nových vzorků vektoru V. Poté je za pomoci dříve naplněné LUT s kosinovými hodnotami vypočteno 64 nových hodnot vektoru V a umístění těchto hodnot do fronty. Po tomto kroku následuje váhování vektoru U oknem, jehož koeficienty jsou dány tabulkou uloženou v paměti. Tím vznikne vektor W a sečtením jeho částí dostaneme 32 výstupních PCM vzorků. Více o této operaci na obrázcích 14 a 15.

7.16 Přehrávání dekódovaného signálu

Po dokončení dekódovacího procesu je nutné výstupní vzorky transformovat na zvukový výstup. Jelikož jsou výstupní hodnoty ve formátu float v intervalu <-1;1>, je nejprve nutné je škálovat. Jak již bylo řečeno, požadujeme výstup ve formátu 16 bitů PCM. Z toho důvodu je nutné všechny výstupní vzorky násobit hodnotou $2^{16}/2$, tj. 32768.

Po této transformaci jsou výstupní PCM vzorky uloženy do paměťové oblasti, která je určena pro řetězené DMA přenosy do audio kodeku AD1835A na vývojovém kitu. Nejnižší vzorkovací frekvence, kterou kodek podporuje je 48 kHz, to je důvod, proč je možné přehrávat pouze soubory s vzorkovací frekvencí 48 kHz. V případě jiných vzorkovacích frekvencí by ještě před odesláním výstupu do kodeku byla nutná operace převzorkování.

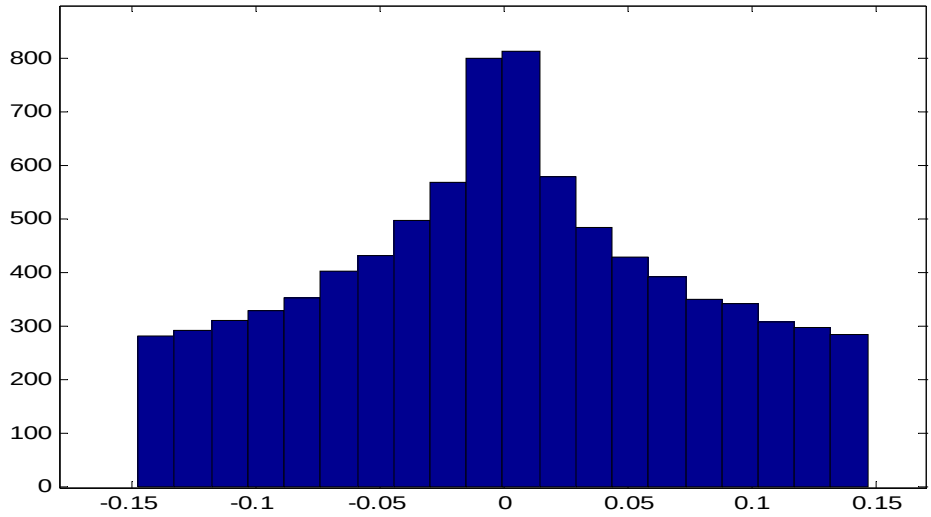
8. Testy

8.1 Testy přesnosti

Pro vývoj vlastního MP3 dekodéru existují oficiální pomůcky. Těmi jsou syntetické testovací soubory. Tyto soubory umožní testovat správnost jednotlivých klíčových operací, jako je dekódování hlavičky, dodatečné informace, Huffmanových bitů atd. Více informací o zdrojových souborech v [20]. Součástí je i zdrojový soubor MP3 se sinusovým signálem 1 kHz a výstupní soubor ve formátu HEX 24 bit PCM, který slouží pro porovnání přesnosti vlastního algoritmu.

Další možností testování přesnosti je porovnání výstupu s implementací v programu Matlab [21]. Tímto způsobem bylo srovnáno 11520 vzorků dekódovaných z Huffmanových bitů (*is*). Díky tomu, že se jedná o celá čísla, oba signály byly naprosto totožné. Při porovnání prvních pěti rámců (11520 vzorků) rekvantizovaných hodnot (*xr*) bylo zjištěno, že implementace v Matlabu a navržený algoritmus se liší ve vypočtených hodnotách rekvantizovaných hodnot o méně než 10^{-7} pro vzorky v intervalu <-1;1>. Tato chyba je dána tím, že implementace v Matlabu pracuje s číselným formátem s dvojí přesností, oproti jednoduché přesnosti algoritmu na DSP.

Navzdory takto přesným výsledků je výsledný zvukový signál vypočten pomocí DSP zatížen chybou v amplitudě až 15 % oproti implementaci v Matlabu. Toto může být způsobeno zaokrouhlováním a nepřesnými hodnotami funkce kosinus během hybridní syntézy a syntézy subpásmových vzorků. Avšak toto zkreslení může být způsoben kteroukoli operací následující po rekvantizaci. Z časových důvodů se nepodařilo zjistit, která operace zatěžuje výsledný signál touto chybou. I přesto je však výstupní signál srozumitelný, ale subjektivně je znatelná zhoršená kvalita oproti originálu.



Obrázek 24 - Histogram odchylek amplitud výstupního signálu v procentech

Obrázek 24 zobrazuje histogram odchylek mezi 8500 výstupních vzorků navrženého algoritmu a implementace v programu Matlab. Test byl proveden na souboru 1000Hz-stereo@192.mp3, který je obsahem přiloženého CD.

8.2 Testy zatížení CPU

Testy byly prováděny pomocí měření na osciloskopu. Začátek a konec sledované operace byl identifikován pomocí změny logické hodnoty pinu procesoru před a po skončení operace. Díky tomu bylo možné měřit časovou náročnost optimalizované verze programu. Optimalizovanou verzi nelze krokovat a tím pádem není možné jednoduše sledovat počet instrukčních cyklů jednotlivých operací. Ladicí verze, kterou bylo možné krokovat

ve vývojovém prostředí, zase vykazovala mnohem horší výsledky a měření zatížení by tím bylo zkreslené.

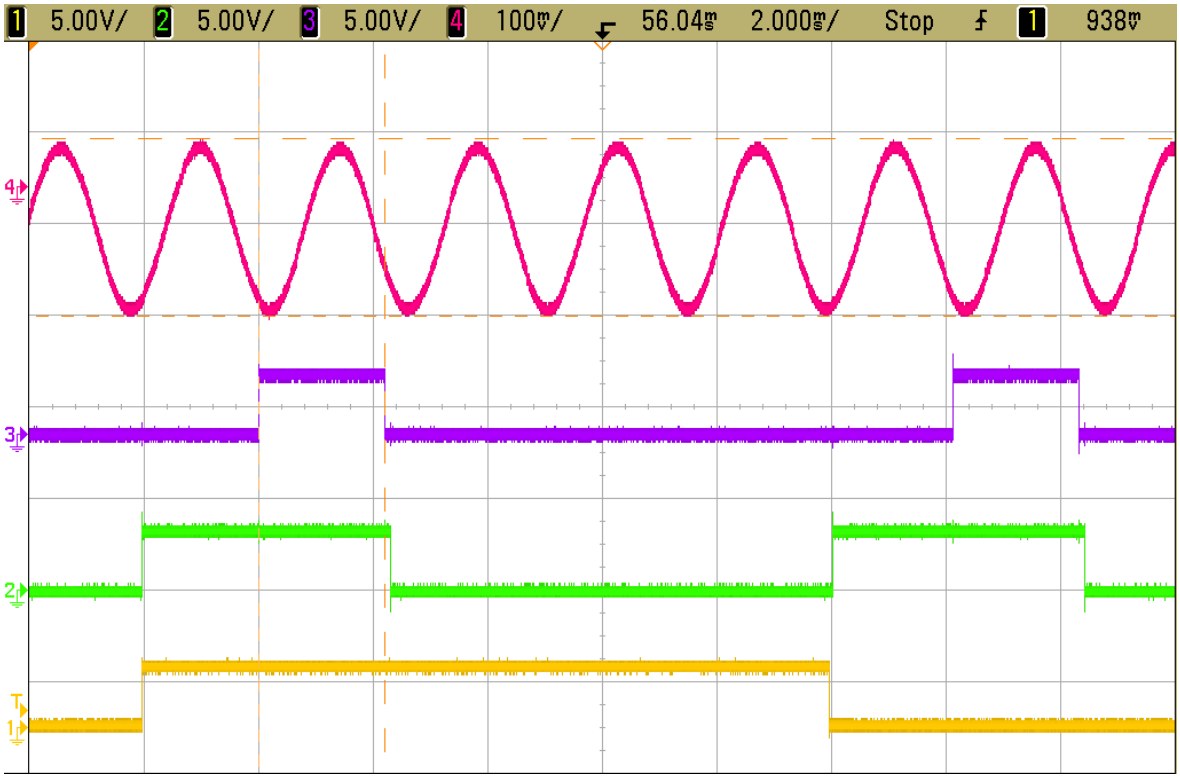
Při prvních testech programu bez optimalizací kompilátoru vyšlo najevo, že procesor není schopen dekódovat MP3 v reálném čase. Maximální čas pro dekódování jedné granule (576 vzorků) je 12 ms. Za tuto dobu je přeneseno všech 576 vzorků do kodeku, který je přehraje se vzorkovací frekvencí 48 kHz. Po zapnutí optimalizací kompilátoru se čas potřebný k dekódování zkrátil na 7,6 ms, ale přesto stále nebylo možné dekódovat vzorky stereo signálu v reálném čase. U stereo signálu totiž potřebujeme za dobu 12 ms provést výpočty pro granuli levého a pravého kanálu (čili máme maximálně 6 ms na jednu granuli). Tento výpočet ale v tomto případě trval 15,2 ms.

Na řadu přišly nutné algoritmické optimalizace. První a nejjednodušší z nich je výpočet vztahů 3.7, respektive 3.10 pro celou granuli a ne pro každý vzorek zvlášť. Žádné zlepšení v rychlosti však nenastalo. Jednalo se o těžko měřitelné zlepšení v hodnotě cca 1,5%.

Další na řadu přišla optimalizace hybridní syntézy, kde dominuje výpočet IMDCT. Referenční zdrojový kód používal kosinové koeficienty z tabulky jen pro dlouhé bloky a pro krátké bloky byla počítána hodnota funkce kosinus přímo. Optimalizace spočívala v použití LUT s kosinovými koeficienty i pro krátké bloky. Toto však nepřineslo stále dostatečné zrychlení algoritmu. Doba potřebná na hybridní syntézu se zkrátila ze 460 na 430 μs, což je zlepšení o méně než 1 %.

Aby bylo možné přehrávat MP3 v reálném čase, přišla na řadu optimalizace tzv. „matrixing“ operací v časově nejnáročnější operaci - syntéze vzorků. Před jakoukoliv algoritmickou optimalizací trvala syntéza 4,88 ms, což je delší doba, než je celkový čas dekódování jedné granule potřebný pro konečnou verzi algoritmu.

Zkrácení celkové doby potřebné pro dekódování o 42 % bylo dosaženo pomocí využití symetrie DCT při „matrixing“ operacích, které probíhají při konečné syntéze. Syntéza je prováděna 18krát pro 32 vzorků a po každé se počítá 64bodová DCT. Využitím symetrie popsané v kapitole 7.15.2 je tedy možné celkovou dobu syntézy zkrátit na polovinu. Praxe ukázala, že 32bodová DCT včetně vytváření 64bodového výsledku za pomoci symetrie trvala o 49,5% kratší dobu (106 μs oproti 210 μs na jednu „matrixing“ operaci). I po této optimalizaci trvá však poslední krok (syntéza subpásmových vzorků) stále asi 50% z celkové doby dekódování jedné granule.



Obrázek 25 - Testy rychlosti algoritmu a výstupní signál

Pozn.: Měřítko časové osy na obrázku 25 je 2 ms na dílek.

Na obrázku 25 můžeme vidět výsledek po všech optimalizacích, kterých bylo dosaženo. Červeně vyznačený dekódovaný signál z audio kodeku. Změna žlutého průběhu vyjadřuje přerušení vyvolané při přenesení celého bloku (2x576 vzorků) do kodeku. Doba mezi přerušeními je 12 ms. Během této doby je nutné provést veškeré výpočty. Logická „1“ zeleného průběhu vyjadřuje dobu zpracování kompletní granule včetně naplnění DMA bloku pro audio kodek. Je vidět, že její kompletní zpracování trvá 4,36 ms. Logická „1“ fialového průběhu vyjadřuju délku syntézy subpásmových vzorků. Průběhy z obrázku 25 byly naměřeny až po algoritmické optimalizaci, a přesto je na nich konečná syntéza vzorků dlouhá 2,2 ms z celkové doby zpracování 4,36 ms. To je stále více, než polovina.

V případě nutnosti dalšího zrychlení by bylo nejlepší výpočet 32bodové DCT v syntéze vzorků transformovat na rychlou DCT a tím by se rapidně snížil čas potřebný k celkovému výpočtu.

9. Závěr

Výsledkem této diplomové práce je dekodér formátu MP3 pro signálový procesor řady SHARC. Navržený algoritmus přehrává v reálném čase soubory s vzorkovací frekvencí 48 kHz a bitrate vyšším, než 192 kb/s. Veškeré funkce a datové struktury potřebné pro dekodér byly navrženy, nebo převzaty z referenčních zdrojových kódů a dále upraveny. Jediným funkčním blokem, který není v algoritmu implementován, je blok dekodéru joint stereo informace. Přesto je algoritmus schopen dekódovat i MP3 v joint stereo módu. Potom však přehrává pouze součtový zvukový kanál.

Správná funkčnost dekodéru byla ověřována pomocí porovnávání výsledků dílčích operací s referenční implementací v programu Matlab. Operace čtení frekvenčních vzorků z datového souboru dává pro oba algoritmy naprosto totožné výsledky. Další operace, rekvantizace vzorků, vykazuje oproti referenční implementaci chybu maximálně 10^{-7}. Výstupní signál je však u navrženého algoritmu zatížen chybou až 15 % v porovnání s implementací v programu Matlab. Původ této chyby se z časových důvodů bohužel nepodařilo zjistit. I přes tuto chybu je zvukový výstup srozumitelný a zkreslení v amplitudě se projevuje jen jako subjektivně horší kvalita dekódovaného zvuku.

Testy zatížení procesoru a vliv algoritmických optimalizací na rychlost zpracování byly prováděny pomocí osciloskopu. Před začátkem a po skončení sledované funkce byla provedena změna logické úrovně vývodu procesoru. Díky tomu bylo možné sledovat dobu potřebnou pro provedení sledované operace. Celkové zatížení procesoru při dekódování dvoukanálového souboru je přibližně 73 %. Vzhledem k vysoké taktovací frekvenci použitého procesoru je to velice vysoká zátěž, avšak podmínka provozu v reálném čase byla splněna.

Při řešení této diplomové práce se podařilo splnit všechny body zadání. Přesto je navržený algoritmus stále vývojovou verzí a nabízí další možnosti paměťové a algoritmické optimalizace. Zejména aplikaci rychlé kosinové transformace na časově náročné operace. Praxe ukázala, že samotná implementace jednotlivých bloků MP3 dekodéru není zas tak složitým problémem. Při vývoji je však velice obtížné kontrolovat správnou funkčnost těchto bloků. Přesto, že známe podobu výstupního signálu, není díky návrhu MPEG-1 layer 3 prakticky možné až do dokončení všech výpočtů určit, zda jsou průběžné výpočty správné, nebo nikoliv. Tato skutečnost byla při řešení zadaného problému největším úskalím. Další problémy způsobovaly mylné informace, týkající se dekódování MP3, v dostupných materiálech.

Navržený algoritmus dokáže dekódovat MP3 soubory bez slyšitelných artefaktů a přehrát je pomocí audio kodeku. V případě reálného nasazení by však bylo nutné nejprve odstranit nepřesnost výpočtu výstupního signálu, následně provést paměťovou a algoritmickou optimalizaci rutiny a nakonec přepsat časově kritické operace do jazyka symbolických adres s ohledem na maximální možné využití výhod architektury SHARC.

Použitá literatura

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

[2] *ID3.org* [online]. 2009 [cit. 2010-08-20]. Developer information. Dostupné z WWW: <http://www.id3.org/Developer_Information>.

[3] RUCKERT, Martin. *Understanding MP3 : syntax, semantics, mathematics, and algorithms*. Wiesbaden : Vieweg, 2005. 247 s.

[4] SUPUROVIC, Predrag. *DataVoyage* [online]. 1998 [cit. 2010-08-20]. MPEG audio frame header. Dostupné z WWW: <http://www.datavoyage.com/mpgscript/mpeghdr.htm>.

[5] *Joint (audio engineering)* [online]. 2010 [cit. 2010-08-20]. Wikipedia. Dostupné z WWW: <http://en.wikipedia.org/wiki/Joint_(audio_engineering)>.

[6] *Lammert Bies - Computer Interfacing* [online]. c2010 [cit. 2010-08-20]. On-line CRC calculation and free library. Dostupné z WWW: <http://www.lammertbies.nl/comm/info/crc-calculation.html>.

[7] SIELER, Martin; SPERSCHNEIDER, Ralph. *MPEG-Layer3 Bitstream Syntax and Decoding* [online]. [s.l.] : [s.n.], 4. 3. 1997 [cit. 2010-08-13]. Dostupné z WWW: <http://www.mp3-tech.org/programmer/docs/bitstream.zip>.

[8] LAGERSTÖM, Krister. *Design and Implementation of an MPEG-1 Layer III Audio Decoder* [online]. [s.l.], 2001. 44 s. Diplomová práce. CHALMERS UNIVERSITY OF TECHNOLOGY. Dostupné z WWW: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.3056&rep=rep1&type=pdf>.

[9] EDSTRÖM, Björn. *Blog.bjrn.se* [online]. 1. 10. 2008 [cit. 2010-08-20]. Let's build an MP3-decoder!. Dostupné z WWW: <http://blog.bjrn.se/2008/10/lets-build-mp3-decoder.html>.

[10] RAISSI, Rassol. *The Theory Behind Mp3* [online]. [s.l.] : [s.n.], December 2002 [cit. 2010-08-20]. Dostupné z WWW: <http://www.mp3-tech.org/programmer/docs/mp3_theory.pdf>.

[11] BOSI, Marina; GOLDBERG, Richard E. *Introduction to digital audio coding and standards*. [s.l.] : Springer, 2003. 434 s.

[12] *Analog Devices* [online]. c2010 [cit. 2010-08-20]. SHARC Processor Architectural Overview. Dostupné z WWW: <http://www.analog.com/en/embedded-processing-dsp/SHARC/processors/SHARC_processor_architectural_overview/fca.html>.

[13] *ADSP-21369 EZ-KIT Lite® Evaluation System Manual* [online]. [s.l.] : [s.n.], September 2009 [cit. 2010-08-20]. Dostupné z WWW:

<http://www.analog.com/static/imported-files/eval_kit_manuals/ADSP-21369%20EZ-KIT%20Lite%20Manual%20Rev%202.2.pdf>.

[14] *Analog Devices* [online]. c2010 [cit. 2010-08-20]. ADSP-21369. Dostupné z WWW: <http://www.analog.com/en/embedded-processing-dsp/sharc/adsp-21369/processors/product.html>.

[15] Underbit Technologies, Inc. *Underbit Technologies, Inc.* [online]. c2010 [cit. 2010-08-20]. MAD: MPEG Audio Decoder. Dostupné z WWW: <http://www.underbit.com/products/mad/>.

[16] *DSPdap* [online]. c2007 [cit. 2010-08-20]. A DSP based Digital Audio (MP3) Player. Dostupné z WWW: <http://dspdap.sourceforge.net/>.

[17] KONSTANTINIDES, Kostas. Fast subband filtering in MPEG audio coding. In *Signal Processing Letters*. [s.l.] : [s.n.], 1994. s. 26-28.

[18] MAROVICH, Scott B. *Faster MPEG-1 Layer III Audio Decoding*. [s.l.] : [s.n.], 2000. 10 s. Dostupné z WWW: <http://www.hpl.hp.com/techreports/2000/HPL-2000-66.html>.

[19] LEE, B.G. FCT - A Fast Cosine Transform. In *IEEE International Conference on Acoustics, Speech and Signal Processing San Diego 1984*. [s.l.] : [s.n.], 1984. s. 28A.3.1-28A3.4.

[20] *MPGedit* [online]. 1994 [cit. 2010-08-20]. MPEG 1 layer 1/2/3 MPEG 2 2/3 Test Data. Dostupné z WWW: <http://mpgedit.org/mpgedit/mpgedit/testdata/mpegdata.html#ISO_m1l2>.

[21] HASSAN, Ahmed. *Matlab Central* [online]. c2010 [cit. 2010-08-20]. Complete Implementation Of a MP3 Decoder. Dostupné z WWW: <http://www.mathworks.com/matlabcentral/fileexchange/27303-complete-implementation-of-a-mp3-decoder>.

[22] *Wikipedia* [online]. c2010 [cit. 2010-08-20]. Codec listening test. Dostupné z WWW: <http://en.wikipedia.org/wiki/Codec_listening_test>.