UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
Odsek za elektrotehniku i računarstvo
Institut za računarstvo i automatiku
Katedra za računarsku tehniku i računarske komunikacije

Jedno rešenje HE AAC za MIPS CPU

Bachelor rad iz oblasti
- DSP I -

Mentor:
Stanislav Očovaj

Student:
Pešut Đorđe E11062

Novi Sad, jun 2008.

SADRŽAJ:

1. Skraćenice

3G2	3GPP file format (file extension)
3GP	3GPP file format (file extension)
3GPP	Third Generation Partnership Project
AAC	Advanced Audio Coding
AAC LC	Low Complexity AAC
ADTS	Audio Data Transport Stream
CRC	Cyclic Cedundancy Check
HDLR	Handler Reference Box
HE-AAC	High Efficiency AAC
IEC	The International Electrotechnical Commission
ISO	The International Organization For Standardization
LATM	Low Overhead MPEG-4 Audio Transport Multiplex
LOAS	Low Overhead Audio Stream
MDAT	Media Data Box
MP4	MPEG-4 file format (file extension)
MPEG-4	Moving Picture Expert Group
PS	Parametric Stereo
SBR	Spectral Band Replication
STSD	Sample Description Box
STSZ	Sample Size Boxes

2. ZADATAK

Realizovati aplikaciju u programskom jeziku C, koja vrši ekstrakciju AAC audio toka iz datoteke u MP4 formatu, i njegovu konverziju u LOAS/LATM, odnosno ADTS format. Ulaz u aplikaciju je datoteka u MP4 formatu, a izlaz iz aplikacije je datoteka u LOAS/LATM, odnosno ADTS formatu.

Realizovana aplikacija treba da zadovolji sledeće kriterijume:

- Da bude napisana u C jeziku (bez C++ ekstenzija)
- Da bude platformski nezavisna (to jest da radi i na little-endian i big-endian platformama)
- Da bude optimizovana u pogledu brzine izvršavanja (dekodiranje AAC toka podataka ne dolazi u obzir!)

Napomene:

- Pri konverziji u LOAS/LATM format koristiti AudioSyncStream() strukturu
- useSameStreamMux treba da ima vrednost 0
- allStreamsSameTimeFraming treba da ima vrednost 1
- numSubFrames treba da ima vrednost 0
- numProgram treba da ima vrednost 0
- numLayer treba da ima vrednost 0
- no_raw_data_blocks_in_frame treba da bude 0
- ne treba koristiti CRC proveru zaglavlja
- audioMuxVersion treba da bude 0

3. Teorijski uvod

MPEG-4 je standard za čuvanje i isporučivanje multimedijalnog sadržaja. Predstavljen je pred kraj 1998. godine. Prvobitno je napravljen kao naslednik MPEG-1 i MPEG-2 standarda. Cilj je bio da se napravi standard za niske bitske brzine, pošto su prethodne verzije već pokrivale visoke bitske brzine, ali je u fazi specfikacije MPEG-4 prerastao u standard koji je imao visok stepen kompresije i podržavao i niske i visoke bitske brzine. [3]

Audio kodovanje u MPEG-4 standardu je urađeno po uzoru na AAC, sistem kodovanja iz MPEG-2 standarda, ali nije kompatibilan sa njim. MPEG-4 je uveo neke novine kako bi poboljšao efikasnost, posebno kod niskih bitskih protoka. AAC je sistem kodovanja sa gubicima. Postoji nekoliko tipova AAC-a, ali su dva najčešća: LC-AAC i HE-AAC. AAC u kombinaciji sa SBR-om i PS-om čini aacPlus. SBR omogućava da audio kodek iporučuje isti kvalitet sa upola manjom bitskom brzinom. PS značajno uvećava efikasnost kodeka za stereo signale sa niskim bitskim brzinama. [3]

Jedna od najvećih snaga MPEG-4 standarda u odnosu na druge je to što je otvoren standard, pa tako svi mogu da imaju uvid u specifikaciju i da prave aplikacije koje će ga podržavati. [3]

4. Opis MP4 formata datoteke

MP4 je 12. deo MPEG-4 standarda. Puštanje fajlova preko mreže je podjednako dobro kao i puštanje na lokalnoj mašini. [2]

MP4 je sastavljen od nekoliko atoma, od kojih neki sadrže i po nekoliko drugih. Na početku svakog atoma je njegova dužina, izražena u bajtima, i njegov tip. Prilikom parsiranja se prvo vidi kog je tipa atom, pa onda ako sadrži neke informacije koje su bitne za dalji rad se parsira, a ako ne sadrži onda se samo preskoči broj bajtova koje pokazuje polje za dužinu tog atoma.

Za konverziju MP4 datoteke u LOAS/LATM I ADTS su potrebni sledeći atomi i informacije koje su smeštene u njima:

- HDLR
- STSD
- STSZ
- MDAT

4.1 HDLR

U ovom atomu se nalazi informacija o tipu *handler*-a sa kojim treba rukovati sadržajem.

Tip atoma je 0x68646c72. 4 bajta koja prethode tipu označavaju dužinu HDLR atoma, a 4 bajta koja dolaze posle tipa, u ovom slučaju, su verzija i neki *flag*-ovi koji su ovde postavljeni na nule. Posle svega ovoga slede polja koja su navedena na slici 4.1.1

```
aligned(8) class HandlerBox extends FullBox('hdlr', version = 0, 0) {
        unsigned int(32)  pre_defined = 0;
        unsigned int(32) handler_type;
        const unsigned int(32)[3]  reserved = 0;
        string name;
}
```

Slika 4.1.1 specifikacija HDLR atoma [2]

Jedina informacija koja se vadi je *handler_type*. Vrednosti koje može sadržati to polje su:

- 'vide' Video track
- 'soun' Audio track
- 'hint' Hint track

Za ovaj rad je potreban samo *soun handler*.

4.2 STSD

U ovom atomu se nalazi informacija o broju kanala i frekvenciji odabiranja.

Tip atoma je 0x73747364. 4 bajta koja prethode tipu označavaju dužinu STSD atoma, a 4 bajta koja dolaze posle tipa, u ovom slučaju, su verzija i neki *flag*-ovi koji su ovde postavljeni na nule. Posle svega ovoga slede polja koja su navedena na slici 4.2.1

```
aligned(8) abstract class SampleEntry (unsigned int(32) format)
 extends Box(format){
        const unsigned int(8)[6] reserved = 0;
         unsigned int(16) data_reference_index;
}

 aligned(8) class SampleDescriptionBox (unsigned int(32) handler_type)
 extends FullBox('stsd', 0, 0){
        int i ;
        unsigned int(32) entry_count;
        for (i = 1 ; i <= entry_count ; i++){
                switch (handler_type){
                        case 'soun': // for audio tracks
                                AudioSampleEntry();
                        break;
                        case 'vide': // for video tracks
                                VisualSampleEntry();
                        break;
                        case 'hint': // Hint track
                                HintSampleEntry();
                        break;
                }
        }
}


class AudioSampleEntry(codingname) extends SampleEntry (codingname){
        const unsigned int(32)[2] reserved = 0;
        template unsigned int(16) channelcount = 2;
        template unsigned int(16) samplesize = 16;
        unsigned int(16) pre_defined = 0;
        const unsigned int(16) reserved = 0 ;
        template unsigned int(32) samplerate = {timescale of media}<<16;
}
```

Slika 4.2.1 specifikacija STSD atoma [2]

Ovde se pročita samo vrednost frekvencije. U ovom atomu se nalazi još jedan atom nazvan ESDS. U njemu se nalaze informacije o broju kanala i o prisutnosti SBR parametra. ESDS atom sadrži nekoliko sekcija, ali najvažnija je peta. Ona počinje bajtom 0x05. Posle njega mogu doći opcioni bajtovi koji mogu imati jednu od sledeće 3 vrednosti: 0x80, 0x81, 0xfe. Iza njih je 1 bajt koji označava dužinu deskriptora, a

posle toga dolaze najvažnija 2 bajta. Poslednji bit prvog je SBR bit, 0 označava da ga nema, a 1 da je prisutan. Ako je SBR prisutan to znači da se prethodno očitana frekvencija deli na pola. Četvrti i treći bit drugog bajta označavaju broj kanala.

4.3 STSZ

U ovom atomu se nalazi informacija o broju i veličini odbiraka. Veličina odbiraka je izražena u bajtima.

Tip atoma je 0x7374737a. 4 bajta koja prethode tipu označavaju dužinu STSZ atoma, a 4 bajta koja dolaze posle tipa, u ovom slučaju, su verzija i neki *flag*-ovi koji su ovde postavljeni na nule. Posle svega ovoga slede polja koja su navedena na slici 4.3.1

```
aligned(8) class SampleSizeBox extends FullBox('stsz', version = 0, 0) {
        unsigned int(32) sample_size;
        unsigned int(32) sample_count;
        if (sample_size==0) {
                for (i=1; i <= sample_count; i++) {
                        unsigned int(32)  entry_size;
                }
        }
}
```

Slika 4.3.1 specifikacija STSZ atoma [2]

4.4 MDAT

U ovom atomu se nalaze podaci enkodovanog AAC-a. Svi su spojeni u jednu celinu, ali se na osnovu podataka o veličini odbiraka mogu podeliti u frejmove.

Tip atoma je 0x6d646174. 4 bajta koja prethode tipu označavaju dužinu MDAT atoma. Ako se dužina svih odbiraka i dužina MDAT atoma ne slažu, onda je potrebno zanemariti prvih 8 bajtova koji su najčešće nule. To se radi jer različiti enkoderi prave različite fajlove. *Nero* ubaci te nule, dok ih *iTunes* ne ubacuje. MDAT je poslednji i najveći atom u MP4 datoteci, i zauzima skoro celu datoteku.

```
aligned(8) class MediaDataBox extends Box('mdat') {
        bit(8) data[];
}
```

Slika 4.4.1 specifikacija MDAT atoma [2]

5. Opis LOAS/LATM formata

LOAS podržava 3 strukture ali se ovde koristi samo *AudioSyncStream()* struktura. Ona sadrži sinhronizacionu reč, multipleksirani element i dužinu frejma. Maksimalna udaljenost dve sinhronizacione reči je 8192 bajta. [1]

```
AudioSyncStream(){
        while (nextbits() == 0x2B7 ) {  /* syncword */          11 bslbf
                audioMuxLengthBytesLast;                        13 uimsbf
                AudioMuxElement( 1 );
                ByteAlign();
        }
}
```

Slika 5.1 specifikacija AudioSyncStream strukture [1]

audioMuxLengthBytesLast – dužina multipleksiranog elementa sa poravnanjem na nivou bajta. [1]

Jedinica koja se nalazi u pozivu AudioMuxElement-a označava da se StreamMuxConfig nalazi unutar tekućeg frejma. [1]

Neke od napomena, koje važe za ovaj zadatak, uprošćavaju rad i mnogobrojne FOR petlje, iz dalje specifikacije, svode na samo jedan prolaz.

Kada se upišu svi biti koji su potrebni za zaglavlje frejma dobija se broj bita koji nije deljiv sa 8, pa se onda bajtovi *payload-a* moraju deliti.

LOAS/LATM se koristi kao standard u *streaming-u.*

6. Opis ADTS formata

```
adts_sequence(){
        while (nextbits()==syncword) {
                adts_frame()
        }
}

adts_frame(){
        byte_alignment()
        adts_fixed_header()
        adts_variable_header()
        adts_error_check()
        for( i=0; i<no_raw_data_blocks_in_frame+1; i++) {
                raw_data_block()
        }
}
```

Slika 6.1 specifikacija ADTS formata [1]

Specifikacija ostalih delova je prikazana u [1].

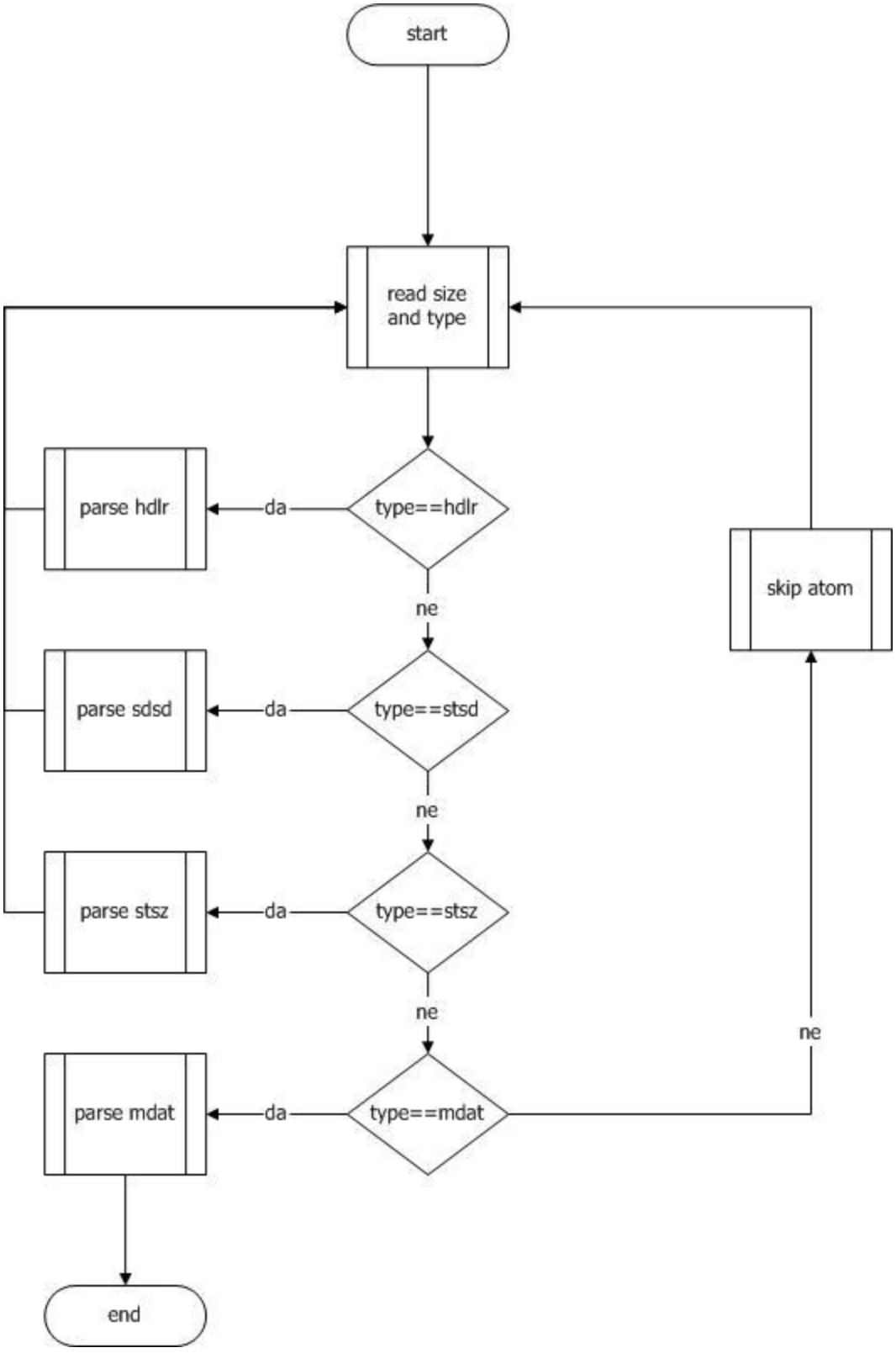
ADTS sadrži 2 tipa zaglavlja, jedno koje se ne menja između frejmova, i jedno koje menja svoje parametre između frejmova.[1]

U ADTS-u broj bita zaglavlja je deljiv sa 8, pa se onda *payload* može samo prekopirati, bajtovi ostaju kakvi su i u MP4 datoteci, odnosno njenom MDAT atomu. Ovde je kao i u LOAS-u olakšan rad sa napomenama iz teksta zadatka.

ADTS se kao i LOAS koristi kao standard u *streaming-u.*

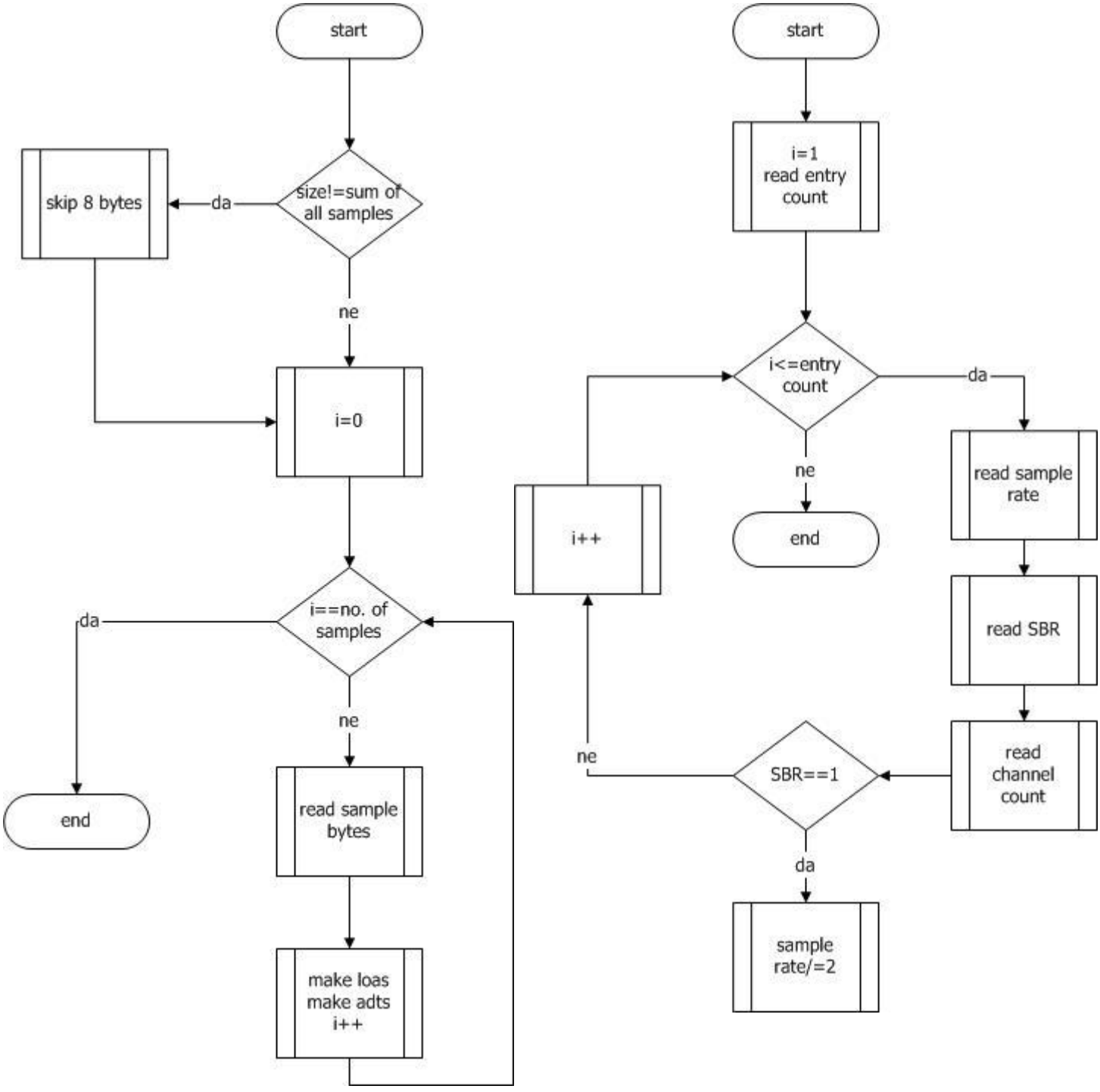
7. Opis rešenja

U zadatku se prvo parsira MP4 datoteka i iz nje se izvuku sve potrebne informacije kako bi se mogle napraviti LOAS i ADTS datoteke.



Slika 7.1 Gloabalni pogled na sistem

Na slici 7.1 se vidi da se iz MP4 datoteke prasiraju samo 4 atoma. Kada se isparsira atom MDAT to je kraj programa, jer je MDAT poslednji atom u MP4 datoteci. Ako se naiđe na neki od atoma koji nije od ova 4 koja su potrebna za parsiranje, program preskače taj atom u MP4 datoteci.



Slika 7.2 Parsiranje MDAT atoma (levo) i parsiranje STSD atoma (desno)

Na slici 7.2 desno, se vidi algoritam za parsiranje STSD atoma. Očitaju se frekvencija i broj kanala i SBR parametar. Ako je SBR parametar aktivan (1), to predstavlja da se prenosi duplo niža frekvencija pa se očitana mora podeliti na pola.

Na slici 7.2 levo, se vdi algoritam parsiranja MDAT atoma. Tu se prvo proveri da li je koder koji je pravio MP4 datoteku dodao nepotrebne nule na početak atoma, pa ako jeste one se onda preskaču. Posle toga se ulazi u petlju koja se vrti onoliko puta koliko ima odbiraka. U svakom prolazu se pročitaju bajtovi odbirka i prave se LOAS i ADTS frejmovi.

Funkcije parsiranja druga dva atoma se svode samo na očitavanje bajtova, pa se algoritmi nisu uvrstili u dokumentaciju.

Iz atoma HDLR se očitava tip *handler-a,* iz atoma STSD se dobijaju vrednost frekvencije sa uračunatim SBR parametrom i broj kanala. STSZ atom sadrži vrednosti veličine odbiraka, a u MDAT atomu su enkodovani podaci koji se izvlače po odbircima.

Pravljenje LOAS frejma se sastoji od upisa odgovarajućih bita u zaglavlje i kopiranje enkodovanih bajtova. ADTS se pravi na isti način samo što se pravi drugačije zaglavlje. Pošto se u zaglavlja pišu biti, a ne bajtovi onda je napravljena posebna funkcija, *writeBits*, koja upisuje zadatu vrednost na određeni broj bita.

Zaglavlje LOAS frejma je takve veličine da nije deljivo sa 8, pa se bajtovi u *payload-u* moraju deliti u odnosu 3 na prema 5. Ti biti se nisu upisivali sa funkcijom writeBits zbog optimizacije koda. Kod ADTS-a nema toga, tamo je sve poravnato.

8. Testiranje

Testiranje je izvršeno uz pomoć *batch* fajla, i test je obuhvatio 61 datoteku sa muzikom. Za sve fajlove test je prošao uspešno. Fajlovi nad kojima se vršilo testiranje, generisani su pomoću Nero i iTunes AAC kodera. Test se sastoji iz sledećih faza:

- Pravljenje referentnog dekodovanog fajla pomoću Coding Technologies AAC dekodera za MP4 fajlove
- Pokretanja programa koji konvertuje MP4 u LOAS i ADTS
- Pravljenje dekodovanog fajla od LOAS-a pomoću Coding Technologies AAC dekodera za LOAS fajlove
- Pravljenje dekodovanog fajla od ADTS-a pomoću Coding Technologies AAC dekodera za ADTS fajlove
- Poređenje dekodovanih fajlova, dekodovani MP4 i dekodovani LOAS, pa dekodovani MP4 i dekodovani ADTS

Testom su obuhvaćeni MP4, M4A, 3GP i 3G2 fajlovi.

9. Zaključak

Prilikom pregledanja MP4 fajlova u heksadecimalnoj predstavi, primetio sam da različiti koderi prave različite fajlove. Nero dodaje neke nule na početak MDAT atoma, dok iTunes to ne radi. Nule se otkrivaju poređenjem veličina MDAT atoma i sume veličina svih odbiraka. Te dve vrednosti bi trebalo da su iste. Ako nisu, onda na početku MDAT atoma postoje nule koje treba zanemariti.

10. LITERATURA

[1] ISO IEC 14496-3 Sub 1
[2] ISO-IEC-14496-12_2005_MP4
[3] MPEG4WhitePaper
[4] ISO_IEC_14496-3 MPEG4 Audio
[5] ISO IEC 14496-3 Sub 4