QDesign Music

A quick analysis

by

Benjamin Larsson

`e-mail:`

banan@student.luth.se

`Version 1.2`

14th January 2004

Contents

1 Abstract

This is an initial analysis of QDesigns audiocodecs QDMC and QDM2 (QDM aka QDesign Music). The goal is to eventually understand the structure of the codecs and finally be able to decode it.

2 Introduction

QDMC is an audio compression-codec from QDesign. It is mainly used in Quicktime [2] from Apple. So far very little is(was) known about this codec. It seams to originally come from Dmitry Shmunk's LB codec. (Low bitrate codec). He was abit active on usenet 96-97, he made the LBpack/LBplay media player. From the history in one of the version of his mediaplayer [4], there is a note of using a Radix-4 FFT implementaion in the player. No other useful information could be found about the technique used for the codec. The only other source of information seams to be from QDesign themself from their pressreleases [1]. The only valuable information from those is that QDMC is a perceptual codec. That indicating a propritary psycoacoustics implementation. Although the pressrelease for QDMC says it is fundamentally different from anything else available at the time (98), one could suspect that this fundamentally difference is in one of the steps of a sub-band codec. And not some other form of approach of compression.

This document will mostly describe QDM2 because the MVP [5] program from QDesign could only encode in that format. Most work is done with 24kbit mono files and that might affect sertain assumptions and facts.

3 Test of QDM2

Testing was done with chirps. The chirp was created with matlab and normalized with a 0.8 factor. A sweep from 0 to 22.05 khz (figure 1) showed that somthing happened around 8kHz. A powerspectrum plot (figure 2) revealed that most of the higher ($> 8kHz$) frequencys are not there. And some strange pattern is repeted 3 times after the drop. If you compare figure 2 with figure 3 striking similarities are found, figure 3 is a powerspectrum plot of a μlaw-converted version of the chirp wavform. The μlaw-conversion is basicly a logaritmic quantization reducing the amount of bits per sample from 16 to 8. This strongly indicates a use of a logaritmic pcm step in the codec.

A chirp test with a higher normalization factor (> 0.9) gave the results shown in figure 4. 6 visible overtones can be found, this indicates some sort of numerical stability, maybe some overflow/rounding error in the encoder. Though some litte trace can be found in figure 2, a close look reveal two lines forming a X.

4 Assumptions

Right now I'm guessing this is a sub-band codec with many conceptual similarities with Layer 2 and 3 of the Mpeg audio codecs. Some use of VLC/RLE/Huffman. Maybe even VQ. IDCT used to resynthesis the sound. QDM2 is the same co-

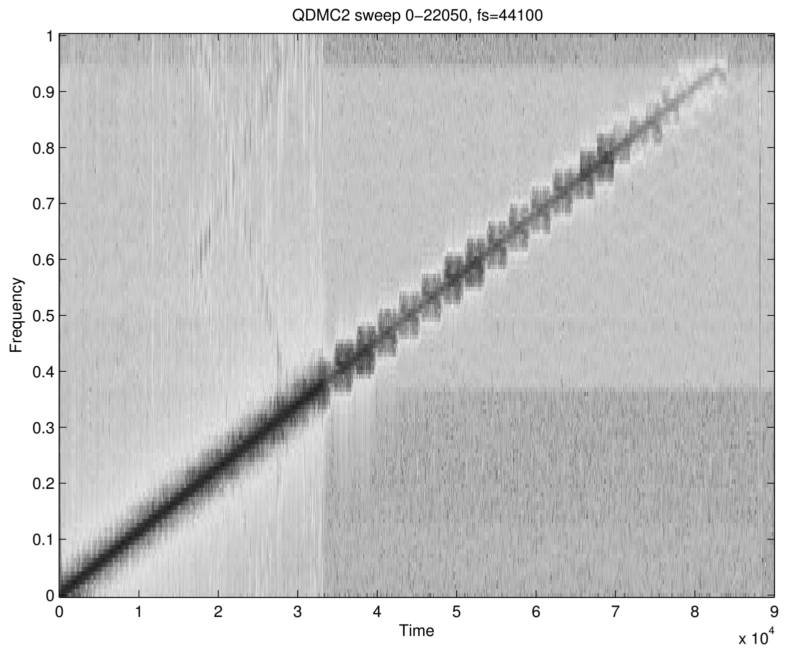

Figure 1: Simple frequencysweep

dec as QDMC but maybe some small changes in the bitstream or the steps to code/decode.

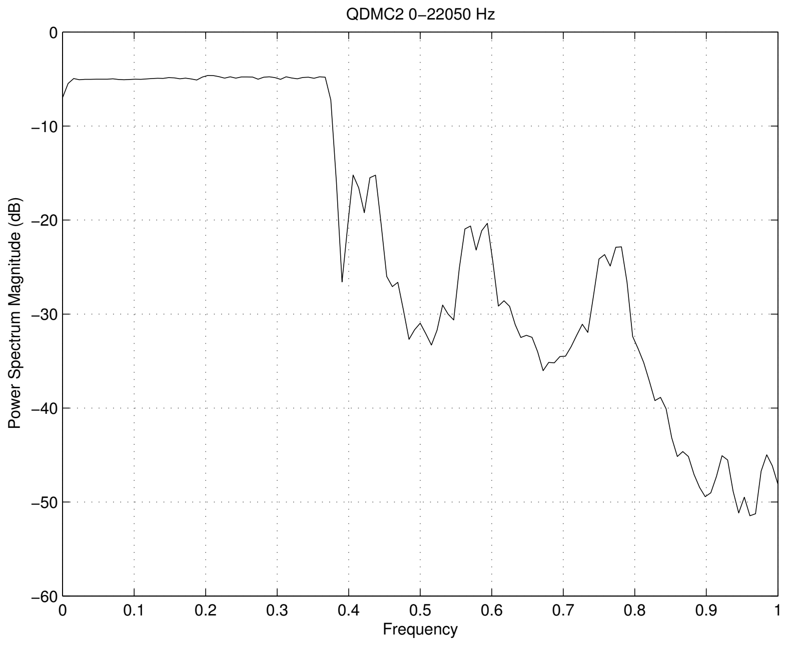

Figure 2: Powerspectrum

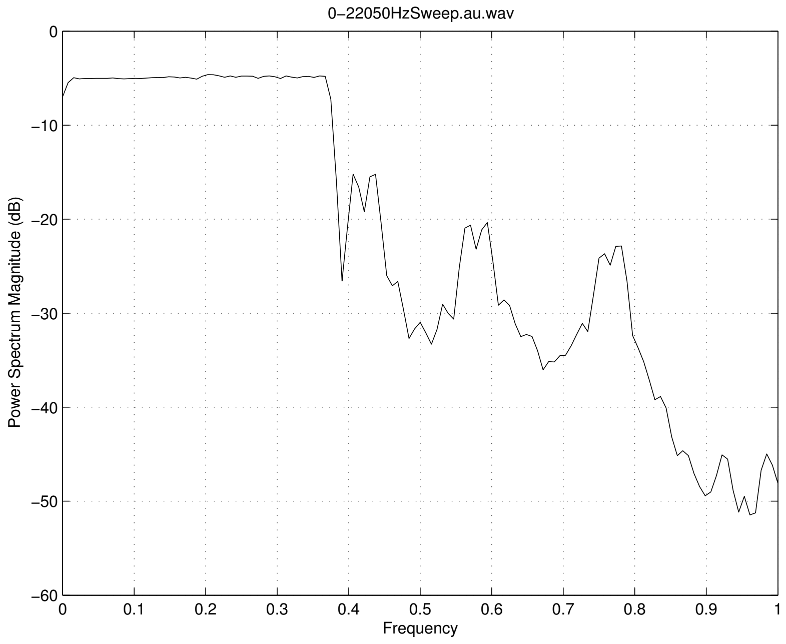

Figure 3: Powerspectrum

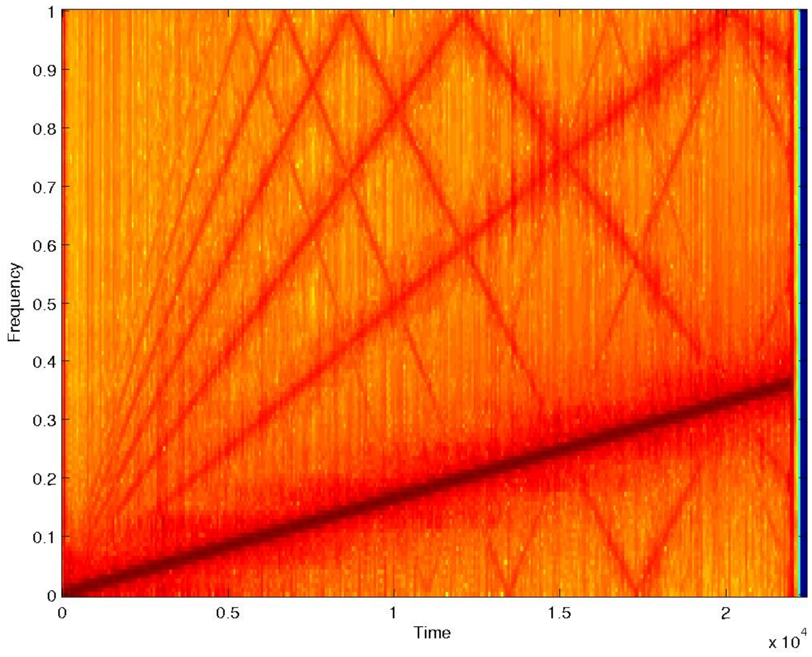

Figure 4: Frequency sweep with overtones

5 Facts

5.1 General

QDM2 resides in a mov container. The container has information about the samplingfrequency and mono/stereo. The codec is a CBR codec.

5.2 Blocktable

The codec works in even blocks and blocksize depends on the samplingfrequency. A block consists of 16bit PCM sampledata. The data that is compressed is simply divided in an even amount of peices.

Samplingfrequency (Hz)	Blocksize (bytes)
8000	1024
11025	1024
16000	2048
22050	2048
24000	2048
32000	4096
44100	4096
48000	4096

Table 1: QDM2 blocksize-table.

Zeropadding is used to fill an even number of blocks.

5.3 Packet table

A compressed block forms a packet accoarding to table 2.

Bitrate (kbits)	Packetsize (bytes)
28	278
48	556
64	742
112	1300
128	1486

Table 2: QDMC2 packet-table.

Mono and stereo data seams to have the same packetsize, this indicating some form of joint-stereo compression.

5.4 Bitstream

A packet is assumed to have a X bytes long header and after that the compressed data. The following has been able to be figured out by looking at hexdumps of the compressed packets. (Table 3.) Table 3 shows the common difference found between files of different bitrate. And by taking the 2nd and 3rd byte and converting it to deciamals, table 4 appears. The numbers in that table is the exact lenght of a packet - 3 bytes. My guess is that the decoder uses this value to know how many more bytes it need to read. With all this information the following headertable is formed. (Table 5)

5.4.1 Checksum calculation routine

Checksum is calculated the following way. Summerize every value in a packet, except byte 4 and byte 5. And the result so the length is a 16bit number. Store the 16bit value divided in byte 4 and byte 5. The following code will calculate the checksum (the packet is represented as a 278 long vector with 8 bits unsigned numbers):

```
Matlab:
checksum = dec2hex(uint16(sum(packetvector(1:128))-sum(packetvector(4:5)))

Pseudo C (I'm no C programmer):

pbyte4 = packet[byte4];
pbyte5 = packet[byte5];

sum = sum(packet);
summinus = sum - (pbyte4 + pbyte5);
checksum = summinus && 0x0000FFFF;
```

The result *checksum* should be byte 4 and byte 5. The order is swapped in QDMC and QDM2.

1	2	3	4	5	6	bitrate (kbit)
82	01	13	XX	XX	09	24
82	02	29	XX	XX	09	48
82	02	E3	XX	XX	09	64
82	05	11	XX	XX	09	112
82	05	CB	XX	XX	09	128

Table 3: QDM2 bitstream-table, the first 6 bytes.

hex	dec
0113	275
0229	553
02E3	739
0511	1297
05CB	1483

Table 4: Hexadecimal to decimal

Byte number in header	Description
1	Identifier, seams to always be 0x82
2,3	Amount of bytes before the packet ends (in hex)
4,5	Checksum value
6	Unknown structure Id value, always seams to be 0x09
7	Lenght of structure
8	Unknown, sometimes it is the same through a whole packet

Table 5: QDM2 headertable.

Samplerate may not be included in the bitstream, I found 24kbit packets with same content (zeros) but from different samplingfrequencys. Indicating that the bitstream does not have any samplingfrequency information. That information can be red from the movcontainer.

5.4.2 Structure decoding

Inside a QDM2 packet there are some structures. Those structures has an id and a length. The procedure to get out all id's and lengths is as follows:

```
set locationpointer to offset of byte 6
Do
save the idvalue (&locationpointer)
save the lengthvalue (&locationpointer + 1)
set locationpointer to locationpointer + lengthvalue +2
Repeat
```

If the idvalue and lenghtvalue is 0 then break the loop.

5.4.3 Bitstream terminator

Bitstreams seams to need 2 NULL bytes to end a packet. Maybe needed for breaking structure calculation.

5.5 Sound resynthesis

The resynthesis from the spatial domain to the temporal domain is done by an iDCT [6]. The iDCT is working with chunks of 64bytes. For 1 packet there are 8 chunks. $8 \cdot 64 = 512$, 512 bytes gets 1024 bytes after μlaw-conversion.

5.6 Decoding procedure

Steps included in the decoding of a QDM2 packet.

- Check checksum
- Decode structures
- Magic (bit shaving, rle, vlc, huffman, etc..)
- iDCT

- *μlaw* to 16-bit pcm

6 Further Progress

More progress can probably be made by looking at hexdumps of packets but most likely dissassembly is a faster way. Proposed way is to make a short movfile with one packet, and then trace what is happening with a debugger. The first step should be some form of unpacking with huffman tables or not.

Under linux the approach with mplayer [3] and gdb would be taken. With mplayer it would be easy to know where in memory the packet is and when access to that memory occurs.

7 Comments

This document is a work in progress and will be updated if/when new information occurs. Most information should be correct but non is guaranteed. If anybody want to help with information just mail it.

8 Version History

1.0 Initial version
1.1 Added checksum calculation algorithm, fixed tyupos and more references
1.2 Added iDCT and structure information

9 References

References

[1] http://www.qdesign.com/news/archives/01_06_98.htm

[2] http://www.apple.com/quicktime

[3] http://www.mplayerhq.hu

[4] ftp://ftp.simtel.net/pub/simtelnet/msdos/sound/lbit157.zip

[5] http://www.rjamorim.com/rrw/qdmc.html

[6] http://sourceforge.net/mailarchive/forum.php?thread_id=3565396&forum_id=9050

10 Hexdump of packets

Just for reference 2 hexdumps are included. Hexcat is used. And by looking at the amount of zeros in the packet of silence it makes sence that a packet with no information is easy to compress. Numbers between [ ] are strucure positions.

10.1 11025 Hz 24 kbit mono white noise

```
00000000 - 82 01 13 66  37[09 21]7c  13 6b 42 02  be 89 0b b8
00000010 - 09 f8 66 06  12 44 e2 9b  d8 73 18 88  6f 62 23 38
00000020 - f2 cd 6b 7a  f1 01 0d 05 [0d 05]53 55  59 16 00[26
00000030 - 01]16[25 1f] 05 1a c8 08  bc fc 2c 49  57 8a 94 0a
00000040 - ba c5 1a 1d  e9 48 5c e9  1b 07 85 14  f5 b2 29 1c
00000050 - 09 91 15[24  42]09 3a a6  8c 65 26 dc  14 11 1f 38
00000060 - 65 64 b8 ac  ab 7d a4 b8  0c a2 00 17  2d 0e 66 3f
00000070 - 82 14 c6 49  0c 0a 98 4c  8b 48 c5 80  2b a6 69 c8
00000080 - 2a a8 12 69  82 09 2a 16  a0 05 0f 88  88 52 11 a6
00000090 - 2b 9b 48 2a  20 e9 01[23  5b]0d e3 2e  ee 70 49 60
000000a0 - 40 51 62 97  63 f0 35 88  0a 47 66 20  96 60 04 61
000000b0 - 83 c8 25 44  64 50 2a 20  58 29 23 40  05 57 d8 10
000000c0 - d9 54 c2 13  e4 30 46 02  54 82 af 33  13 42 5c e4
000000d0 - 51 42 88 44  2e 61 62 1d  82 1a 40 08  51 30 04 13
000000e0 - 0b 22 93 62  93 7b cc 60  19 09 a0 13  88 94 08 22
000000f0 - b8 a6 31 0a [22 1e]b5 92  f0 cd ca 52  d7 5a eb df
00000100 - 6e c9 b0 a6  b5 0a 96 15  2f 27 a4 84  e7 f5 ca ae
00000110 - 94 d5 fa 01 [00 00]
```

10.2 11025 Hz 24 kbit mono silence

```
00000000 - 82 01 13 0c  da[09 13]cc  2f 93 f9 65  32 bf 4c e6
00000010 - 97 c9 fc 32  99 5f 26 f3  cb 04[0d 05] 00 00 00 00
00000020 - 00[26 01]16 [25 01]55[24  01]55[23 01] 20[22 01]00
00000030 -[00 00]00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000040 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000050 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000060 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000070 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000080 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000090 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
000000a0 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
000000b0 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
000000c0 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
000000d0 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
000000e0 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
000000f0 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000100 - 00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00000110 - 00 00 00 00  00 00
```