PLUSV SPECIFICATION

"VLSI Solution PlusV"

Revision History			
Rev.	**Date**	**Author**	**Description**
1.0	2001-10-12	HH	The official, first public release.
0.8	2001-10-10	HH	MP3 interleaving redone.
0.72	2001-09-04	HH	PO release.
0.7	2001-08-29	HH	Changed order inside FDB.
0.6	2001-08-29	HH	Allowed wrapping for lSinFreq.
0.5	2001-08-28	HH	Delay issues addressed properly.
0.4	2001-08-27	HH	Slight changes to sine, all concepts and bits ought to be there now.
0.01	2001-07-12	HH	Initial documentation.

Contents

List of Figures

1 Abstract

Traditional lossy compression formats like MP3 are very good at compressing data that has high correlation. However, at high frequencies, where most of the energy consists of noise, these compression standards spend lots of bits trying to represent accurately noise waveforms, which essentially cannot be compressed properly.

In this document, a completely new audio compression enhancement decoder format, called PlusV, is presented. PlusV is not a stand-alone compressing format. Instead, PlusV uses an existing encoder to handle the lower half of the available bandwidth while using specialized bit-saving techniques for encoding the higher half of the audio spectrum. PlusV can compress the upper half of the bandwidth of an audio file typically with slightly less than 8 kbit/s. Then, the lower half of the bandwidth can be sent to an already existing traditional encoder, which now has a much easier task in compressing the rest of the bandwidth. After compression has finished, the bitstreams of the traditional encoder and PlusV may be combined. It is also possible to create bitstreams that can both be played with a traditional decoder with lower quality, and with a PlusV aware decoder with full quality.

With PlusV, a 44.1 kHz near-CD quality stereo bitstream can be created with 64 kbits/s, and a 32 kHz moderate quality stereo bitstream can be created with as little as 48 kbit/s. For mono sound, even lower bitrates may be acheved. This makes PlusV a perfect choice for portable MP3 players, or streaming applications with limited bandwidth.

PlusV doesn't require much code nor working RAM memory, and it requires only moderate amounts of processing power to encode and decode. Decoding a 64 kbit/s MP3+V will typically require less processing power than decoding an ordinary 128 kbit/s MP3.

PlusV has been more or less under development since late 1999, and now the time seems right to publish the format for the public.

The intent is that according to this document, VLSI Solution will present a hardware encoder and decoder based on an already-existing VLSI Solution MP3 decoder. Our hopes are that with the help of example source code and this document, also other software and hardware developers will take into their use this open and pretty much free[1] format.

[1] For details of how you are allowed to use this format, refer to Chapter 9.

1.1 References

1. ISO/IEC Standard 11172-3
2. ISO/IEC Standard 13818-3
3. http://www.plusv.org/
4. http://www.lame.org/
5. http://www.vlsi.fi/

1.2 Definitions

% Modulus operator with consistently working negative numbers.
Examples: 56 % 48 = 8, -7 % 48 = 41.

B Byte, 8 bits.

b Bit.

dB decibel, the logarithmic unit for measuring ratios. For convenience, the definition of dB is slightly misused in this document. Where the ratio 2:1 actually is 6.0206 dB, this ratio is referred to as 6 dB. Thus, 96 dB does not mean a ratio of 63096:1, but exactly 65536:1.

FDB Fat Data Block = absolute block.

fs Sampling frequency.

IC Integrated Circuit.

K Computer kilo (1024).

k SI kilo (1000).

LDB Lean Data Block = difference block.

M Computer mega (1024*1024 = 1048576) when combined with bits (b) or bytes (B). Otherwise SI Mega (1000000).

PlusV VLSI Solution's format for extending current codecs.

VS_DSP VLSI Solution's DSP core.

VS_DSP^2 VLSI Solution's DSP core with some basic peripherals.

1.3 Conventions

As a default, all numbers are in decimal format. Hexadecimal numbers are marked with a leading “0x”, like “0x1234” (4660), and binary numbers are marked with a leading “0b”, like “0b11001010” (202).

Bitstreams are always presented MSB first, and first byte first.

1.4 Patents

VLSI Solution has a patent pending for PlusV technology.

1.5 Summary

An intruduction to PlusV and its ideology is given in Chapter 2.

Basic concepts for PlusV and the dataflow of the format are gone through in Chapter 3.

The bitstream format of PlusV is explained in Chapter 4.

The way PlusV should be played back is introduced in Chapter 5.

PlusV compliance requirements are handled in Chapter 6.

Chapter 7 tells how to connect PlusV with MP3 files to create complete MP3+V files.

Chapter 8 contains our standard disclaimer.

The PlusV license is presented in Chapter 9.

Finally, VLSI Solution’s contact information is shown in Chapter 10.

2 Introduction

PlusV is an add-on for existing lossy audio compression schemes. With PlusV, it is possible to create bitstreams that are up to one half smaller than otherwise. This is achieved by using some ear weaknesses with high voices, and by thus compressing the upper half of the audio bandwidth with radically different methods from existing codecs.

This document will present everything needed to write a PlusV decoder, and encoder. Encoder functionality is briefly presented, followed by the definition of a decoder. A working encoder is defined as any encoder that produces a proper bitstream defined in this document. A working decoder will be more strictly defined.

Although this document does not contain detailed information on how to make a PlusV encoder, the encoding process is presented below to show why certain things exist in the decoding process. The process for encoding a PlusV bitstream goes as follows.

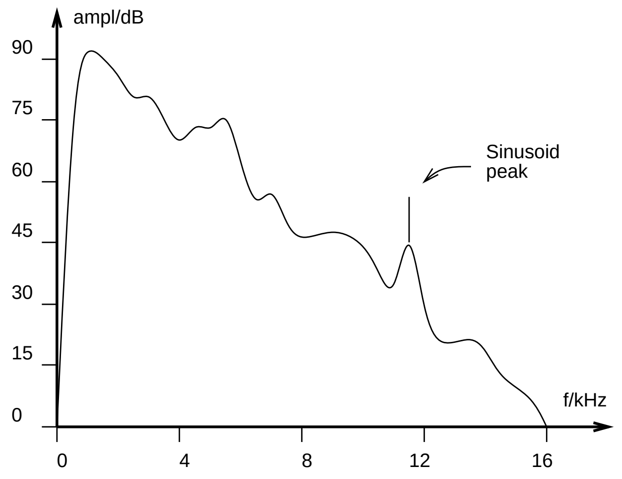

Figure 1: Original Signal Spectrum

In Figure 1 a typical signal spectrum is presented. In this example picture one peak frequency has been placed at 11.5 kHz for demonstration.

This original signal is divided by a half-band filter to two different bands, both holding exactly half of the complete bandwidth.

In Figure 2 you can see the low-pass filtered portion of the signal. Because there are no signals above fs/4, this signal may and will be decimated by two, to half of the original sample rate. For instance, for a 44100 Hz signal, the low-pass filtered and decimated signal would be 22050 Hz.

The downsampled signal is then sent to the conventional compression engine, such as Lame for MP3 encoding.

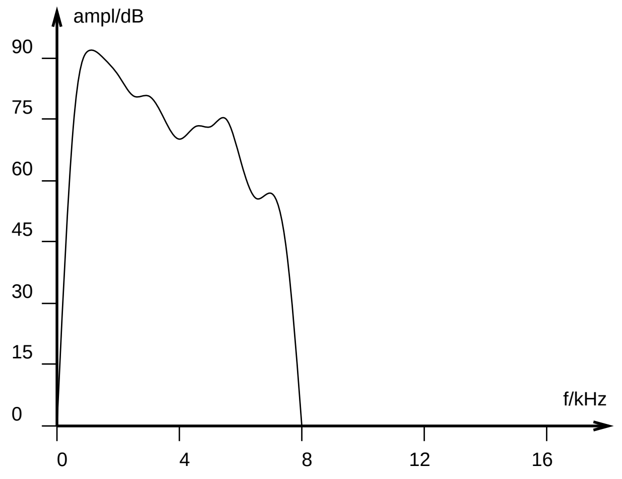

Figure 2: Low-Pass Filtered Signal Spectrum

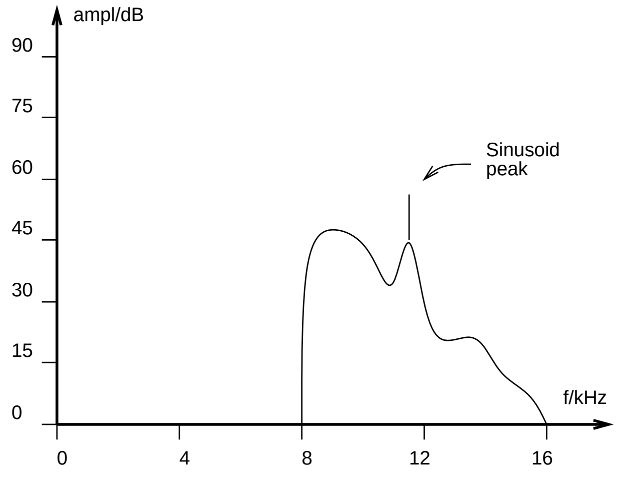

Figure 3: High-Pass Filtered Signal Spectrum

In Figure 3 the high-pass filtered half of the signal is presented.

Typically the highest octave of a signal contains only noise. However, every now and then a harmonic or synthesizer frequency actually goes upto this range. Thus, upto four different sinusoidal signals may be stored and later reproduced. If such signals are found, like the example's 11.5 kHz signal peak, their energies and frequencies are stored to the PlusV file, and their energies are removed from the original signal.

In Figure 4, the 11.5 kHz sine peak has been removed, and now it is assumed that the audio spectrum consists solely of white noise. The frequency spectrum is divided into eight bands, and the energy of each band is calculated and stored to the PlusV bitstream.

Now the PlusV bitstream is combined with the bitstream that was compressed

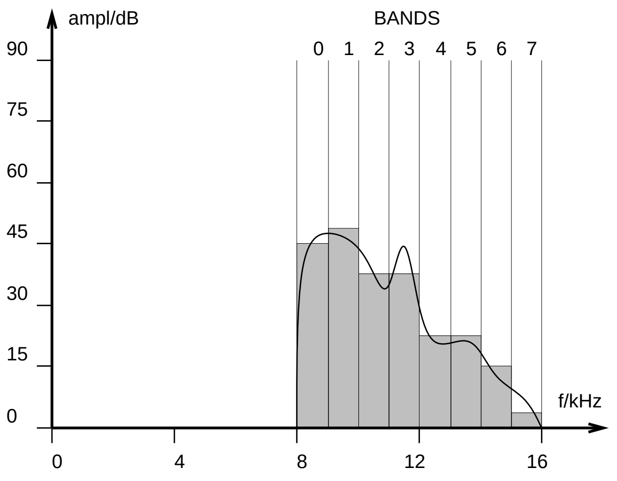

Figure 4: High-Pass Filtered Signal Spectrum After Sin Removal

with the conventional encoder. The way this is achieved depends on the the audio format. Details for how to combine the bitstreams for Mp3 and PlusV are presented in Chapter 7.

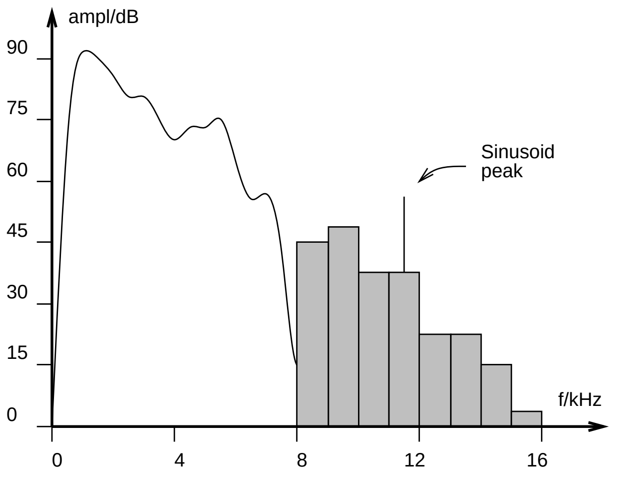

Figure 5: Reconstructed Signal Spectrum

In Figure 5, the signal has been restored. First the low-pass filtered signal that has been encoded to a bitstream format (like MP3) is decoded and by this way the lower half of the spectrum is restored. Then, bandwidth limited noise generators are used to reconstruct the noise parts of the upper half of the frequency spectrum. Finally, sinusoidal signals are added.

While the idea of noise and sine generators may seem very harsh and simplistic, it is in many cases very difficult for the average user to tell any difference between the original signal and the reconstructed, 64 kbit/s signal. As of writing this, there are still some audio files where significant differences can be heard, but as PlusV encoders start to develop, these initial problems will probably vanish soon.

3 Concepts

With a PlusV encoder, audio data is downsampled to 1:2 of the original sampling rate. For instance, for a 44100 Hz CD, the playback rate is 22050 Hz. This will give the user a bandwidth of slightly below 11 kHz. This downsampled data is fed to a conventional audio encoder, like Lame for the MP3 format. For more information of how to combine PlusV with MP3, see Chapter 7.

At the same time the PlusV encoder stores high-frequency components in its own, very low-bitrate format. The both data formats are then combined in a way dependent of the original codec format and limitations.

When decoding, the data and signal path presented in Figure 6 is used (the example figure is for MP3+V, but the image is also applicable to other formats).

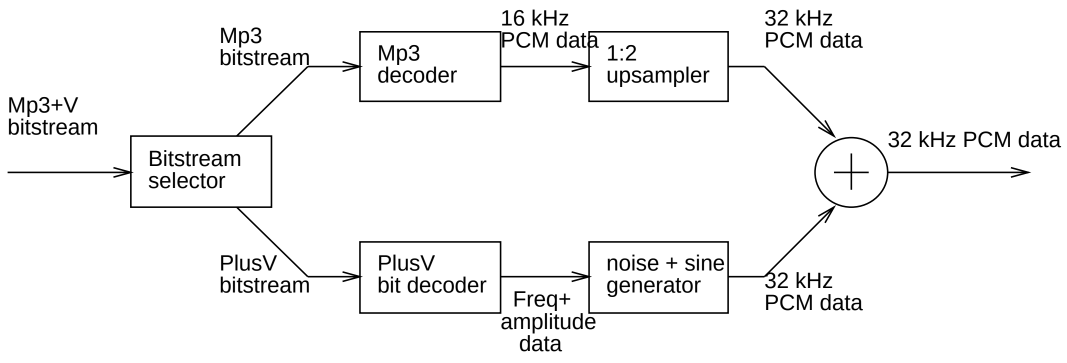

Figure 6: Example MP3+V Decoder Data Path

4 The Bitstream

PlusV is a block-oriented format. It is generally a good idea to choose a PlusV blocksize that is similar to the format that is used to put PlusV on top.

4.1 Different Data Blocks

There are two kinds of data blocks in PlusV: Fat Data Blocks (FDB) and Lean Data Blocks (LDB). The idea in this is that FDBs contain a proper, non-ambiguous header, basic data for the format options, and absolute values for audio data. LDBs contain highly compressed difference audio data, and may be used most of the time. However, LDBs are not safe landing points if one starts playback from the middle of a file.

It is required that music files contain some FDBs so as to allow a user to jump in the middle of a song and still expect to get the PlusV enhancements almost immediately. Also, the very first PlusV block of any file should always be an FDB. The decoder is only allowed to start decoding PlusV data from an FDB. If even for once the LDB data seems to be corrupt (lCounter value is wrong or lBlockLen is less than 3), or the LDB is not located exactly where expected, the decoder is required to stop decoding PlusV information and not to start before the next FDB is encountered.

4.1.1 Fat Data Block

The Fat Data Block consists of the bitfields presented in Figure 7. The number before the name of the bitfield tells how many bits it occupies. An optional number after the bitfield tells how many times the field will be repeated.

$fHeader$ is a non-ambiguous header that is used for recognizing that there is PlusV information to follow. The four byte-values are 0xd6, 0x53, 0xd6, 0xfd.

$fVersion$ is the version number of the format. Currently the number is 0.

$fBlockLen$ is the total length of the Fat Data Block in bytes, including the four fHeader bytes. If fChannels is greater than 2, fBlockLen is multiplied with fChannels and divided by two with normal integer downwards truncation. Note, that it is allowed to use fBlockLen that is larger than the area actually used for PlusV data, as long as the rest of the block is padded with zeros. By this way it is easier to create constant bit-rate streams.

There are eight noise bands, with frequency ranges from fs/4 + fs/4*n/8 to fs/4 + fs/4*(n+1)/8 -1, where n = 0..7. However, of these only seven lowest may be used with PlusV. $fBands$ is the number of lowest bands that actually are used for

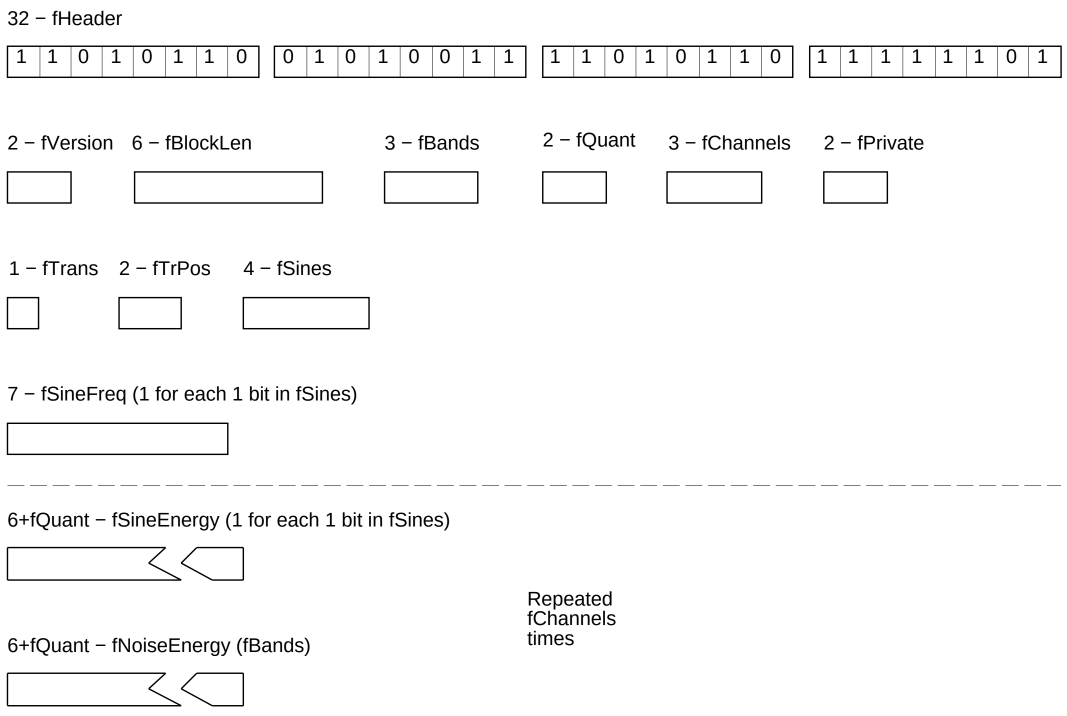

Figure 7: Fat Block Structure

audio. With a 32 kHz sampling rate, a 14 kHz bandwidth can be reached by using all seven available bands, while 6 or 5 bands are sufficient for 44100 and 48000 Hz, respectively, to gain a 19+ kHz bandwidth.

fQuant tells which quantization table (see Chapter 4.2) will be used for decoding fields in Lean Data Blocks. fQuant may change during the file to make it easier to implement constant bit-rate applications. The following table is used to decode the value of fQuant (note, that value three is reserved for future extensions). The Modulo field tells which number to use to modulate the value to get a correct remainder. Examples: If the value is -7 and Modulo is 48, the final value will be 41. If the value is 56 and Modulo is 48, the final value will be 8.

fQuant	**Table**	**See Chapter**	**Modulo**	**RMult**
0	1	4.2.1	48	4.0 dB
1	2	4.2.2	96	2.0 dB
2	3	4.2.3	192	1.0 dB
3	N/A	N/A	N/A	N/A

fTrans tells if the block is a transition block (1) or not (0). For details about how transition blocks differ from normal blocks, see Chapter 5.1.

fTrPos is provided only if fTrans = 1. This field will tell which quarter of the compression block to use for the transition.

fChannels tells how many channels there are in the PlusV bitstream. This value should be the same as for the main bitstream, and may not change during the file.

fPrivate may be used for private purposes for that particular encoder. Note, that these bits are not meant to be wildly used by the user, but to be predetermined before PlusV data is embedded within another format, like MP3.

fSines is a bitmap of four bits, and it tells which sine generators are active. For instance, a pattern of '1011' tells that sine generators 3, 1 and 0 are active.

A *fSineFreq* is repeated for each '1' bit of fSines, starting with the left-hand (i.e. highest valued) bit. fSizeFreq is to be decoded to a frequency of (128+fSizeFreq)/512*fs. These frequencies are common to all audio channels.

All of the following fields are repeated fChannels times:

A *fSineEnergy* is repeated for each '1' bit of fSines, starting with the left-hand (i.e. highest valued) bit. Each sine which's frequency was defined with a corresponding fSineFreq is to have the energy of (fSineEnergy*RMult-188) dB, where 0 dB is a level where the energy is 1/32 = 0.0312 (= amplitude is 1/4 of the maximum). A more practical way of saying this is that the amplitude of the sine is (fSineEnergy*RMult-188)/2 dB where 0 dB is set the the same sine as above. The maximum allowed value for fSineEnergy is 192/RMult-1.

fNoiseEnergy is repeated fBands times. Each band is numbered from 0 upwards, and the decoder must create a noise signal that has a spectrum of fs/4 + fs/4*n/8 to fs/4 + fs/4*(n+1)/8 -1. The total energy for the noise is (fNoiseEnergy*RMult-188) dB, where 0 dB is a level where the energy is equal to a sine signal with energy of 1/32 (= amplitude is 1/4 of maximum). The value 0 is interpreted as mute, i.e. zero power. The maximum allowed value is 192/RMult-1.

4.1.2 Lean Data Block

The Lean Data Block is much more compressed than the relatively lavish FDB. The biggest difference is that whereas all values were presented to LDB as absolute values, LDB values for energies and frequencies are delta values, i.e. they only modify the old value.

The LDB consists of the bitfields presented in Figure 8.

lCounter is a 2-bit counter that is always cleared when an FDB is encountered. Each time an LDB is found, this counter is added to by 1 (the value for the first LDB after an FDB is 1). Value 3 is forbidden, thus after 2 comes 0. If the decoder encounters an illegal ICounter value, it must assume the bitstream is broken and no longer contains PlusV data. Thus, all energy values must be cleared to zero. This state must be maintained until the next FDB is encountered.

lBlockLen is the total length of the Lean Data Block in bytes. It is functionally

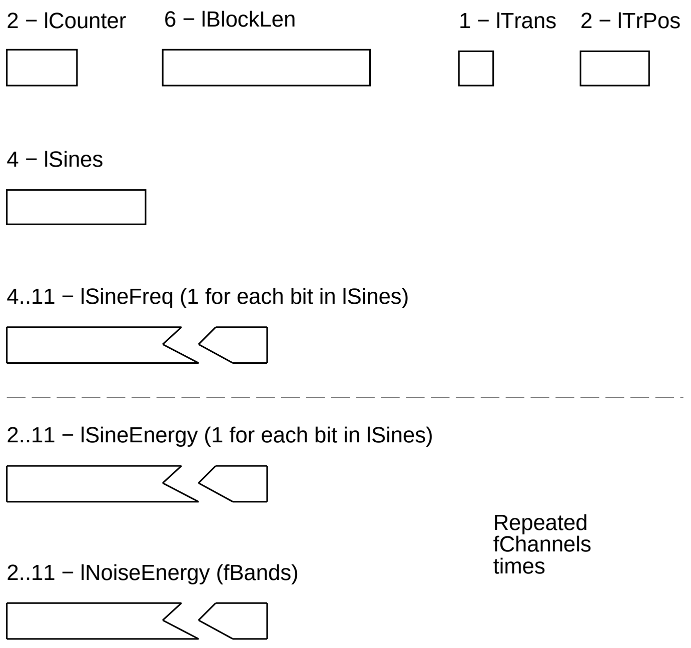
2 – lCounter
6 – lBlockLen
1 – lTrans
2 – lTrPos
4 – lSines
4..11 – lSineFreq (1 for each bit in lSines)
2..11 – lSineEnergy (1 for each bit in lSines)
Repeated fChannels times
2..11 – lNoiseEnergy (fBands)

Figure 8: Lean Block Structure

the same as fBlockLen.

lTrans, *lTrPos* and *lSines* are functionally similar to *fTrans*, *fTrPos* and *fSines*, respectively.

lSineFreq is repeated just like fSineFreq except the base to repeat it is lSines instead of fSines. Also, the new value is not an absolute value, but a difference to the previous value, i.e. newFreq = (oldFreq + lSineFreq) % 128. The resulting value is decoded using Bit-Code Table 3 regardless of fQuant. However, as seen in the formula the Modulo value to be used is not 192 as for energy data, but 128. Thus, in some cases there are more than one way to get from one frequency to another. An example would be a change from 10 to 105, which can be achieved either by a change of +95, or -33. Note, that if there are two ways to meet the same value, their bit length is always the same (11 bits).

The following fields are repeated fChannels times:

lSineEnergy is repeated just like fSineEnergy, except the base to repeat them is lSines instead of fSines. The decoding table for the field is selected with fQuant. The difference with fSineEnergy is that instead of an absolute value, the decoded value is used as a difference to the previous value, i.e. newEnergy = (oldEnergy + lSineEnergy) % Modulo.

lNoiseEnergy is repeated for fBands times, just like fNoiseEnergy. lNoiseEnergy has their values modified like lSineEnergy, including wrapping.

4.2 Bit-Code Tables

4.2.1 Bit-Code Table 1 (RMult = 4 dB)

Prefix	Code	Value
1	0	0
	1	-1
01	0	1
	1	-2
001	00	2
	01	3
	10	-4
	11	-3
0001	000	4
	001	5
	010	6
	011	7
	100	-8
	101	-7
	110	-6
	111	-5

Prefix	Code	Value
0000	00000	8
	00001	9
	00010	10
	00011	11
	...	...
	01101	21
	01110	22
	01111	23
	10000	-24
	10001	-23
	10010	-22
	10011	-21
	...	...
	11101	-11
	11110	-10
	11111	-9

4.2.2 Bit-Code Table 2 (RMult = 2 dB)

Prefix	Code	Value
1	00	0
	01	1
	10	-2
	11	-1
01	00	2
	01	3
	10	-4
	11	-3
001	000	4
	001	5
	010	6
	011	7
	100	-5
	101	-6
	110	-7
	111	-8

Prefix	Code	Value
0001	0000	8
	0001	9
	0010	10
	0011	11
	0100	12
	0101	13
	0110	14
	0111	15
	1000	-16
	1001	-15
	1010	-14
	1011	-13
	1100	-12
	1101	-11
	1110	-10
	1111	-9

Prefix	Code	Value
0000	000000	16
	000001	17
	000010	18
	000011	19
	...	...
	011101	45
	011110	46
	011111	47
	100000	-48
	100001	-47
	100010	-46
	100011	-45
	...	...
	111101	-19
	111110	-18
	111111	-17

4.2.3 Bit-Code Table 3 (RMult = 1 dB)

Prefix	Code	Value
1	000	0
	001	1
	010	2
	011	3
	100	-4
	101	-3
	110	-2
	111	-1
01	000	4
	001	5
	010	6
	011	7
	100	-8
	101	-7
	110	-6
	111	-5
001	0000	8
	0001	9
	0010	10
	0011	11
	0100	12
	0101	13
	0110	14
	0111	15
	1000	-16
	1001	-15
	1010	-14
	1011	-13
	1100	-12
	1101	-11
	1110	-10
	1111	-9
0001	00000	16
	00001	17
	00010	18
	00011	19
	00100	20
	00101	21
	00110	22
	00111	23
	01000	24
	01001	25
	01010	26
	01011	27
	01100	28
	01101	29
	01110	30
	01111	31
	10000	-32
	10001	-31
	10010	-30
	10011	-29
	10100	-28
	10101	-27
	10110	-26
	10111	-25
	11000	-24
	11001	-23
	11010	-22
	11011	-21
	11100	-20
	11101	-19
	11110	-18
	11111	-17
0000	0000000	32
	0000001	33
	0000010	34
	0000011	35
	...	...
	0111101	93
	0111110	94
	0111111	95
	1000000	-96
	1000001	-95
	1000010	-94
	1000011	-93
	...	...
	1111100	-36
	1111101	-35
	1111110	-34
	1111111	-33

5 Block Playback Implementation

5.1 Changes Inside Blocks

When block parameters change, the new values mustn's simply override the old ones. This would lead to audible clicks and pops at block length intervals. To avoid this, all changes must be done gradually during the whole block as shown in Figures 9 and 10. The gradual changes should be implemented for all the following fields: fNoiseEnergy, fSineFreq, fSineEnergy, lNoiseEnergy, lSineFreq, lSineEnergy. The only exception to this is that if Sines(x) has been 0 in the previous block and 1 in the current block, SineFreq is set immediately to the new value.

However, there are changes that are much faster than the approx 12 ms that one block takes. Thus transient blocks were added to the format. A transient block is otherwise as a normal block, but the whole change takes place in just on fourth of the block length. This allows the format to have transient blocks that have only a 2-bit overhead over a normal block.

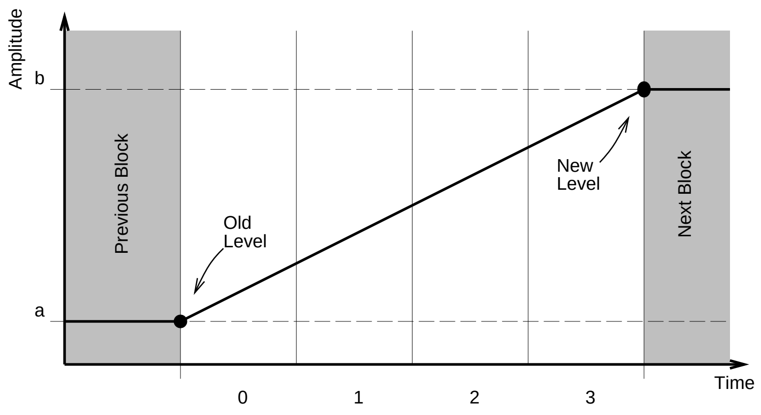

Figure 9: Signal Change in Normal Block

Figure 9 shows how to change amplitude for a sine signal during a normal block. Note, that although amplitude change is shown here as linear, it is also in-spec to change energy or amplitude in a non-linear fashion. The recommended method is, however, to change amplitude linearly (in linear scale, not dB scale).

If a block is a transient block, the change must be implemented as shown in Figure 10. Here the whole change is done during one fourth of the block, and fTrPos or lTrPos defines which fourth (0..3) is the one for change. For the rest of the block, no changes are made.

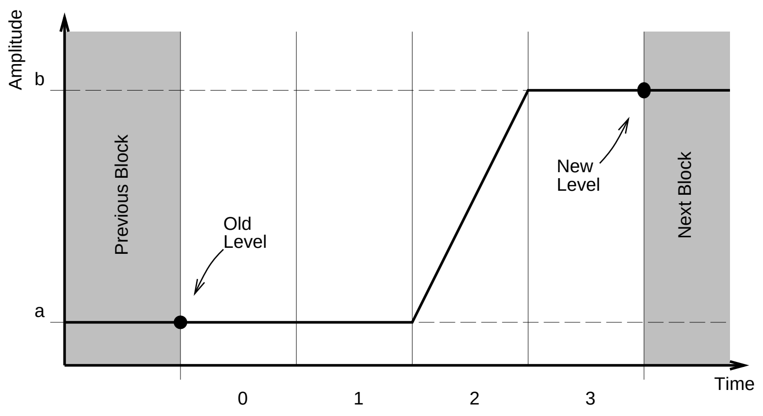

Figure 10: Signal Change in Transition Block, fTrPos/lTrPos=2

5.2 Parameters that May Change During a Bitstream

Parameters that may be defined only at the beginning of the stream and that may not be changed during the stream are: fVersion, fChannels.

Parameters that may change whenever an FDB is encountered are: fBands, fQuant, and fPrivate.

All other parameters may change in any block.

5.3 Remembering Old Values

5.3.1 What to Remember with an FDB

All old values are discarded. SineFreq is set to 0 for all the frequencies not transmitted in the FDB. NoiseEnergy is set to 0 for all the bands that are not in use. This way an FDB makes it absolutely sure that the decoder is in sync with the encoder.

5.3.2 What to Remember with an LDB

When an LDB is encountered, the following values are modified: NoiseEnergy, SineFreq, SineEnergy.

If the Sines(x) bit for sine x is 0, the sine amplitude is zeroed, but the sine frequency for the audio generator is left as-is. This way the value may be used in a subsequent sine block even though the sine generator was not in use for a while.

6 Decoder Compliance

For a decoder to be compliant with PlusV, it has to meet the following rules.

NOTE! THIS CHAPTER IS STILL NOT FINAL, AND IT WILL BE REWRITTEN IN A LATER EDITION OF THIS DOCUMENT!

6.1 Upsampling

The upsampling filter used to interpolate the low-pass filtered data must have at least a 5.5 dB attenuation at fs/4, and 80 dB attenuation at 0.3*fs. After that point, it must always be at least -80 dB. Passband ripple may not exceed 0.1 dB.

6.2 Sine Generation

Sine generation power must be within 1% of that specified, and the frequency must be with 0.5% of that specified.

6.3 Noise Generation

Noise generation power must be within 2% of that specified for all bands.

6.4 Delay

The system delay of the decoder may not be more than that specified in the specific decoder standard + 256 samples. See Chapter 7 for details of MP3+V. The delay difference for the lower and upper half of the bandwidth (i.e. the basic compression and PlusV parts of the signal) may not be more than 16 samples.

7 MP3+V

7.1 General

With MP3 1.0 sample rates (48, 44.1 and 32 kHz), one compression frame lasts for 1152 stereo samples (two granules, each 576 samples). With MP3 2.0 sample rates (24, 22.05 and 16 kHz), one compression frame lasts for 576 stereo samples (one granule). With PlusV, this 576 sample frame corresponds to 1152 samples in reconstructed, sample-doubled audio. Thus, it was decided that one PlusV compression block should consist of a number that is something that may be divided from 1152.

Because sample rates vary so much in the decoded samples (from 32 to 48 kHz), it was decided that two different PlusV decode block lengths would be allowed, namely 576 and 384 samples. This way the normal block length can be slightly above 10 ms, and a transition block can make its transition in approx. 3 ms, which is enough to fool the ear to think that a change has happened instantaneously.

It is recommended to use 576 sample blocks for 48 and 44.1 kHz, and 384 sample blocks for 32 kHz. It is, though, allowed not to follow the recommendation if audio quality reasons dictate otherwise.

PlusV blocks are always meant to be decoded with the corresponding MP3 frame in the stream (see Chapter 7.3.2).

After each PlusV block, there are 0..7 zero bits to pad the following block (be it PlusV or MP3) to be byte-aligned.

7.2 Using fPrivate Bits

Bit 1 of *fPrivateBits* is unused, and should always be set to 0.

If Bit 0 if *fPrivateBits* is 0, then the length of the PlusV block is 576 samples and there are always two PlusV blocks for each MP3 block. If the bit is 1, the length of a block is 384 samples and there are always three PlusV blocks for each MP3 block. This value may change during the bitstream.

7.3 Embedding PlusV to an MP3 Bitstream

7.3.1 Building a PlusV Frame

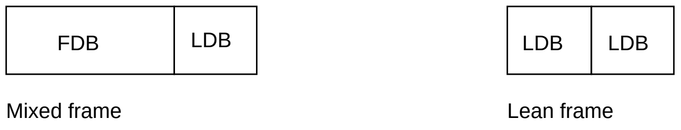

Figure 11: PlusV Frame Types, fPrivate = 0

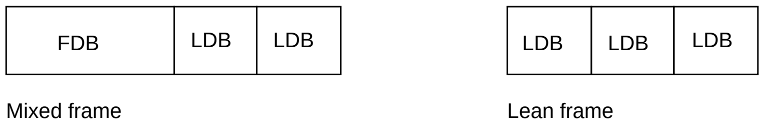

Figure 12: PlusV Frame Types, fPrivate = 1

Depending on the value of fPrivate (see Chapter 7.2), either two or three PlusV blocks are combined to create a PlusV frame (see Figures 11 and 12). The first PlusV frame in a file must always be a Mixed frame. Other frames can be either Mixed or Lean frames, as long as there are enough Mixed frames to satisfy the requirements in Chapter 7.4.

7.3.2 Location of PlusV Frame Inside MP3

Understanding this chapter requires some knowledge of how MP3 streams work. The definitive reference for this is the ISO/IEC 11172-3 standard, and even there, particularly Figures A.7.a and A.7.b.

MP3 allows for a maximum of 255 bytes of internal buffering for its 2.0 sampling rates 16000, 22050 and 24000 Hz. This buffering is demonstrated in Figure 13. As you can see, parts of the following “main info X” frame data may be buffered to the previous MP3 frame. This information is coded to the main_data_begin field in the side info data.

In the example Figure 13, part of the main info data for frame 2 is buffered to frame 1. The main_data_begin pointer in the side info data for frame 2 contains the information for where to start decode frame 2. After all of the sync and side info data for frame 2 have been decoded, the main info data parts for frame 2 are combined (i.e. the sync+side info data are removed from the middle of frame 2 data). Then frame 2 can be decoded properly.

Now, how do we embed PlusV data in a compatible way to this bitstream?

First, a PlusV frame is built from two or three PlusV blocks as described in Chapter 7.3.1. The, the last octet of the MP3 frame is searched for. (Note: if

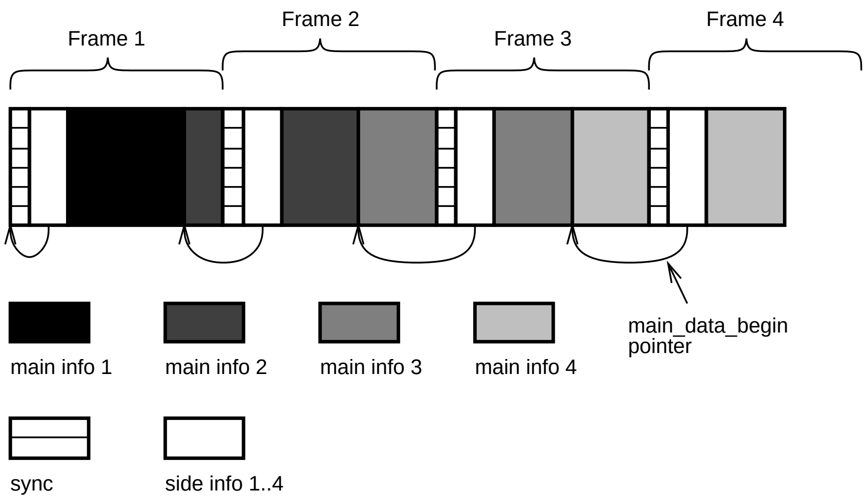

Figure 13: MP3 Bitstream Organization

PlusV is a builtin part of an MP3+V encoder, this information is fairly easy to get by.) The PlusV information is encoded startind from the beginning of the next octet.

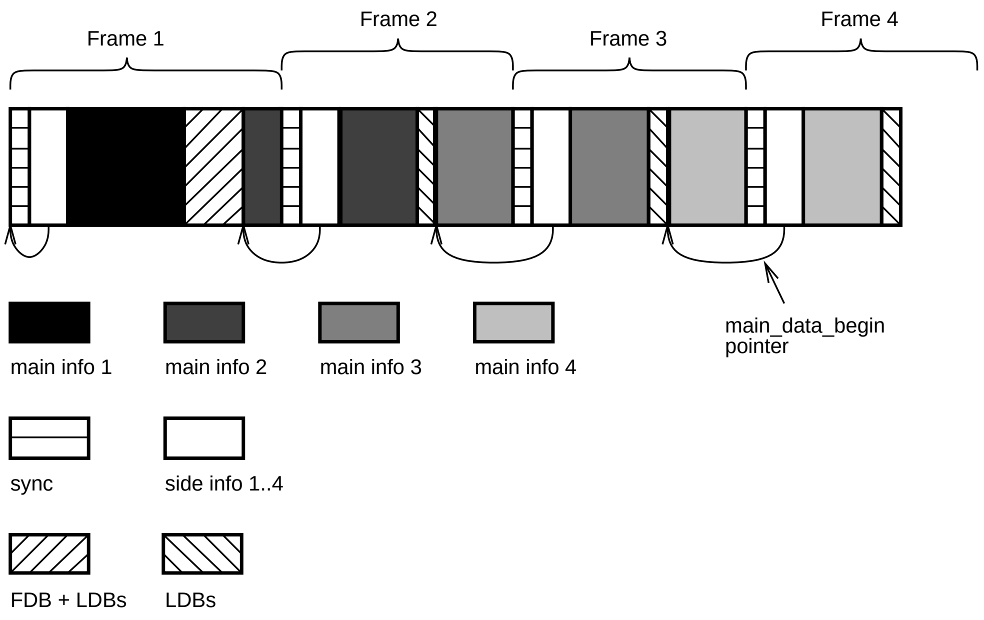

Figure 14: MP3+V Bitstream Organization

An example of where to code PlusV frames is given in Figure 14. A conventional MP3 decoder never expects to find any kind of useful data in this area, because the next MP3 frame's main_data_begin pointer points to the beginning of the next main info data instead of the PlusV frame. Actually, this kind of skipping data is used in normal MP3 encoders when the bitrate is so high that frames are not filled. Thus, this is way of embedding MP3 data is 100% downwards compatible with existing MP3 decoders.

7.3.3 Matching Decoder Delays

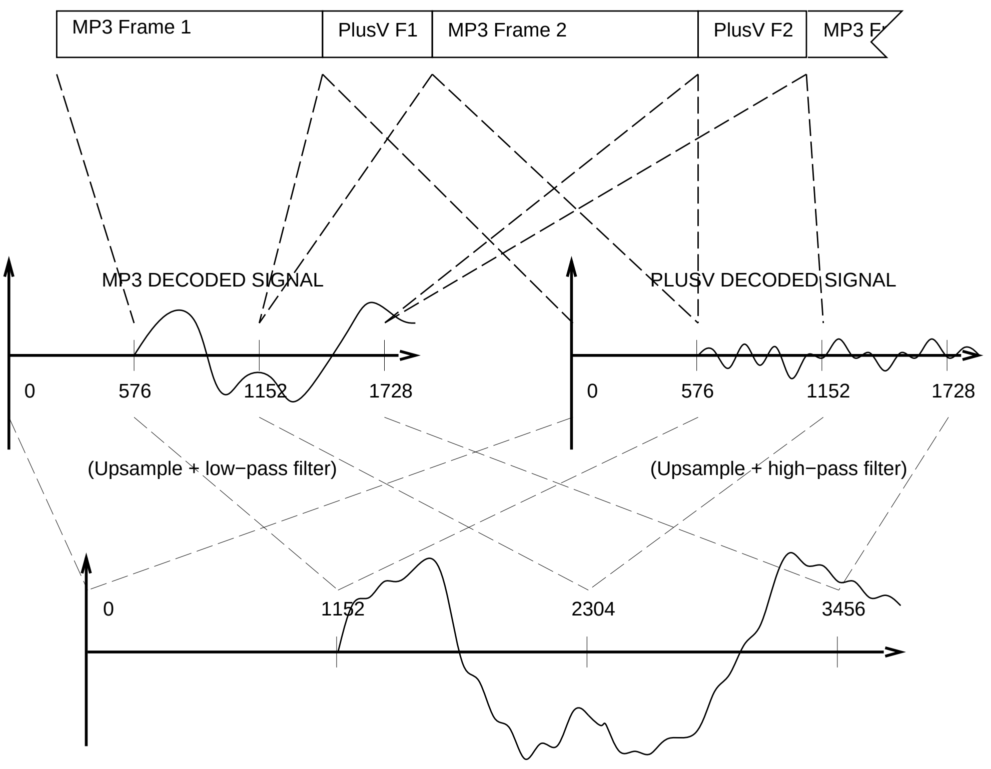

Figure 15: MP3 and PlusV Delay Matching

There is an inherent one frame, i.e. 576 sample delay in MPEG 2 layer 3 decoding because of filtering. However, PlusV doesn't have such filter delays (see Figure 15). To make implementing decoders easier, it was decided that this imparity would be compensated by in the encoder. Thus, it is the job of the encoder to expect that the decoder has an additional 1152 sample delay for MP3 data in the not-yet-downsampled domain, i.e. the 32..48 kHz domain. In other words, the encoder should delay its input to PlusV by an extra 1152 samples before downsampling or 576 samples after downsampling.

In practise this means that the first two or three PlusV blocks (depending on fPrivate) should always be empty because of the delay.

If a MP3+V decoder has a delay that is not 576 samples it has to take special measures to compensate for its non-standard behaviour.

7.4 Minimum Number of FDBs

At least every 66th PlusV block must be an FDB. The upper limit for FDBs is every 2nd and every 3rd for fPrivate bit 0 values 0 and 1, respectively. FDBs don't need to be at constant intervals. If the bitstream to be encoded is put to noisy lines where transmission errors are likely to happen, it is recommended to use a more FDBs than the minimum.

Of the two or three PlusV blocks preceding an MP3 block, only the first one may be an FDB.

8 Disclaimer

Every effort has been made to make this document as reliable and dependable as possible. Before publishing the first version of this document, an independent decoder has been written in addition to the reference decoder.

Nevertheless, normal restrictions apply. VLSI Solution takes no responsibility in the rightness or usefulness of any data in this document.

This means that VLSI Solution OY absolutely does not take any responsibility whatsoever if this format doesn't work or destroys something.

9 License

You may add PlusV support freely to any open, non-proprietary audio format, such as MP3, Ogg Vorbis, etc. It is not allowed to add PlusV or ideas gotten from PlusV to any closed format, for which there is no specification available to the public. It is absolutely not allowed to change the PlusV format in such a way that it breaks existing PlusV applications. There is no use of a 5% or 10% better compressed PlusV variant if it breaks the old players!

PlusV may be used free of charge, and products containing PlusV may be sold. The only exception are hardware codecs, like MP3 decoders. For them, a moderate license fee must be payed to VLSI Solution OY. Also software products must be reported to VLSI Solution OY through the web pages in http://www.plusv.org/ for a free license.

Any software product that uses PlusV or tehnology derived from PlusV, must have the following text in it's main "help" or "Creators" page: "PlusV audio enhancement technology provided by VLSI Solution OY". If the product don't have a "help" or "Creators" page, the text must be clearly visible in associated documentation.

10 Contact Information

VLSI Solution Oy
Hermiankatu 6-8 C
FIN-33720 Tampere
FINLAND

Fax: +358-3-316 5220
Phone: +358-3-316 5230
Email: plusv@vlsi.fi
URL: http://www.plusv.org/
URL: http://www.vlsi.fi/