VS1053B PATCHES AND FLAC DECODER

VSMPG "VLSI Solution Audio Decoder"

Project Code:	VS1053
Project Name:	Support

Revision History

Rev.	Date	Author	Description
2.5	2017-11-17	POj	Encoding mode fix.
2.4	2016-06-23	POj	Experimental DSD64 decoding (.dff and .dsf). Mono output mode extended to allow selecting left or right channel only.
2.3	2015-05-25	POj	More accurate VU meter, scale is now 1 dB steps.
2.2	2015-01-02	POj	Application hook at 0x60 to disable zero-sample insertion.
2.1	2013-09-26	PO	ID3v2 parser added. You see 0x4944 in HDAT1 while data is being skipped.
2.1α	2013-09-11	PO	Resampler to be able to use rate tune with 48kHz and 12.288MHz XTALI (not with FLAC). Rate tune is now more accurate with large values. Pause mode also in parametric_x.reserved[2] bit 1 (compatible with vs1063a).
2.00	2013-07-03	PO	AAC: Simplified LATM/LOAS parser (for DAB+).
1.98	2013-02-05	PO	AAC: HE-AAC: PS initialization bug fixed.
1.96	2012-11-23	PO	AAC: MP4 less prone to get stuck on invalid atom sizes.
1.95	2011-11-16	PO	AAC: MP4 StreamDiscard missing. (Fixed) FLAC now sets DO_NOT_JUMP during header decode.
1.9	2011-10-24	PO	AAC: ADTS decoding ignored the crc flag. (Fixed)
1.8	2011-04-28	PO	Sometimes right channel of Ogg Vorbis not played if left channel is empty (Fixed).
1.71	2011-04-21	PO	Bit reservoir check cleared a register causing erroneous detection of Ogg Vorbis (Fixed).
1.7	2011-02-18	PO	parametric_x.reserved[2] bit 0 sets mono output mode.
1.61	2011-02-02	PO	AAC: PNS fix correction (was only active for 1000 transition frames).

Contents

List of Figures

1 Description

There are some known problems in the VS1053b firmware that this patch addresses.

- Vorbis: Occasional windowing overflow.
- Vorbis: Ogg streams that have the highest bit set in stream serial number are not played.
- Vorbis: Ogg streams that have larger than 32-bit sample position will skip some samples every 2^{32} samples (27 hours at 44.1 kHz).
- Vorbis: If the first four bytes of Ogg Vorbis file wrap from the end of the stream buffer to the beginning, the decoding will get stuck.
- Vorbis: If left channel is empty (digital 0), right channel may be ignored.
- AAC: MP4 format with unused data at the start of the mdat atom are not played.
- AAC: HE-AAC: PS header frame must be present in the first encountered SBR frame when PS is present or the decoding may crash.
- AAC: Automatic clock change and feature drop is too forgiving.
- MP2: Invalid MPEG header can cause a lockup.
- IMA ADPCM encoding does not send encoded data.
- SS_REFERENCE_SEL is cleared by firmware so the higher 1.65V reference voltage can not be permanently activated.
- SS_DO_NOT_JUMP is not by default cleared at software reset.
- MP3: mp3 decoder in vs1053b does not calculate the amount of valid data in bit reservoir, so if you start decoding in a middle of an mp3 stream, decoding can access invalid data and cause audible disturbances.
- Samplerate is changed immediately although audio buffer can still contain data with the old samplerate.
- AAC: Automatic feature drop does not work with non-implicit upsample mode.
- AAC: PNS of left channel can be corrupted in transition frames.
- AAC: ADTS decoding ignored the crc flag.
- AAC: MP4 header decoding did not StreamDiscard() at the correct point.
- AAC: MP4 header decoding sometimes assumed an atom had subatoms when it did not.
- AAC: Added a simplified LATM/LOAS parser for DAB+ use.
- Resampler: to be able to use rate tune with 48kHz and 12.288MHz XTALI.
- ID3v2 can contain data that matches a decodable format. An ID3 parser now skips the extra data correctly.

When the patch is loaded, it starts automatically (writes 0x0300 to AIADDR) and restarts the system (you should wait for DREQ to rise). The patch must be re-loaded after each hardware or software reset. If you replace software reset by writing 0x0300 to AIADDR, you do not need to reload the patch.

File	IRAM	Description
vs1053b-patches.plg	0x50..0x2b5, 0x300..0x5f3	compressed plugin
vs1053b-patches.c	0x50..0x2b5, 0x300..0x5f3	old array format
vs1053b-patches-latm.plg	0x50..0x7f4	with LATM/LOAS
vs1053b-patches-latm.c	0x50..0x7f4	with LATM/LOAS
vs1053b-patches-flac.plg	0x50..0xf3e	with FLAC decoding
vs1053b-patches-flac.c	0x50..0xf3e	with FLAC decoding
vs1053b-patches-flac-latm.plg	0x50..0xff6	with FLAC and LATM/LOAS
vs1053b-patches-flac-latm.c	0x50..0xff6	with FLAC and LATM/LOAS
vs1053b-patches-dsd.plg	0x50..0xf3c	with DSD64 decoding
vs1053b-patches-dsd.c	0x50..0xf3c	with DSD64 decoding

These patches use the application address to start automatically (the last entry in the patch tables writes to SCI_AIADDR), but do not use it afterwards.

The second version contains a simplified LATM/LOAS parser for LC-AAC and HE-AAC (DAB+), the third one contains a FLAC decoder, and the fourth one both the FLAC (minimal implementation) and the LATM/LOAS parser (this does not include the ID3v2 parser).

Both old loading tables and the new compressed plugin format is available. The new plugin format is recommended, because it saves data space and future plugins, patches, and application will be using the new format.

Unless otherwise specified, this patch is **not** compatible with other vs1053b plugins, especially the VS1053 Ogg Vorbis Encoder Application: you need to give a software reset and load the right code when you switch between decoder and encoder.

Specifically VS1053 AD Mixer and VS1053 PCM Mixer do work with the basic version (with of without LATM/LOAS, but without FLAC) as long as you load and start the vs1053b patches package first.

1.1 Fixes in Detail

Vorbis: Occasional windowing overflow

Occasional windowing overflow which could cause audible clicks is eliminated.

Ogg: High bit of stream serial number

Ogg stream serial number is ignored, so Ogg streams that have the highest bit set in stream serial number are now played.

Ogg: 64-bit sample position

Without the patch Ogg streams skip some samples each time the 64-bit sample position crosses the 32-bit boundary (every 2^{32} samples, i.e. 27 hours at 44.1 kHz). This no longer occurs with the patch.

Ogg: Ogg Vorbis file starts at the end of stream buffer

If the first 4 bytes of the Ogg Vorbis file wrap in the stream FIFO, the decoding gets stuck (caused by the stream read pointer escaping the stream buffer. This patch fixes the code.

Vorbis: Empty left channel may cause right channel to be ignored

If left channel was empty (digital zero, i.e. not sent at all) in mid/side stereo format, the right channel could also be ignored. The patch fixes this problem.

AAC: MP4 format

Unused data at the start of the **mdat** atom is now skipped so the start of audio data is located correctly.

HE-AAC: PS header

Switch from SBR without PS to SBR with PS crashed the decoding unless a PS header frame was present in the first encountered SBR frame. This patch fixes the problem and PS is correctly initialized.

AAC: Clock Change and Feature Drop

User-specified clock adder allows the decoders in vs1053 to increase internal clock if they encounter data that is not correctly decodable with the base clock. If the internal clock is not fast enough even with the clock adder active, decoding features are dropped. In the case of HE-AAC, first the parametric stereo (PS) decoding is dropped, then SBR decoding is run in downsampled mode, and if these are not enough, SBR decoding is dropped completely.

In VS1053b the clock change and feature drop routine is a bit too forgiving of audio buffer underruns, and underruns can continue to happen peridiocally without the routine dropping decoding features. This causes small clicks on the audio (audio buffer underruns).

Also, if you use the non-implicit upsample mode, the feature drop does not work at all.

This patch makes the routine less tolerant and quicker to drop features. This limits the audible disturbances in the decoded audio to one or two per song. Also, feature drop now works even when you use the non-implicit upsample mode.

AAC: PNS and Transition Frames

Pseudo-random noise substitution (PNS) information can get corrupted when reordering spectral lines for transition frames (8 short frames). Only the left channel is affected.

This patch fixes the problem.

AAC: ADTS Decoding with CRC

ADTS format decoding ignored the crc protection flag. So, when crc field was present in the audio, the field was not correctly skipped. This could also crash the decoder.

This patch fixes the problem.

AAC: MP4 Header Decoding Missing StreamDiscard

MP4 format header decoding did not discard the already read stream at the correct point. This could cause data to be read before it appeared into the stream buffer and “mdat” header being missed, thus no AAC decoding.

This patch fixes the problem.

Lockup if MP2/MP1 with MPEG2.5

MPEG2.5 extension is only valid with layer III decoding, but an invalid header can have MPEG2.5 active with layer I and layer II as well. This combination can cause the decoder to try to read a larger frame than what the SDI buffer can hold, causing it wait for enough data indefinitely. There are some safe-guards against this in the data handling routines, but they seem to be insufficient in this case.

This patch correctly ignores this invalid MPEG header combination, so the lockup condition is not triggered.

SS_REFERENCE_SEL

The vs1053b firmware clears the 1.65V reference select bit at each volume set operation. This makes the higher reference hard to use.

SCI_STATUS bit 8 is now used as an additional reference select bit, which is copied to bit 0 by this patch. Set both bits to the state you want for better future compatibility.

Note that when you use the higher reference voltage, AVDD must be between 3.3 V and 3.6 V.

To activate the higher reference voltage after reset, wait that analog powerdown has been turned off (SCI_STATUS becomes 0x0040), then write 0x0141 to SCI_STATUS.

SS_DO_NOT_JUMP

The vs1053b firmware does not clear the DO_NOT_JUMP bit from SCI_STATUS during software reset, so sometimes fast forward and rewind appear to be forbidden unnecessarily. This patch clears the bit at startup (and when you write 0x300 to SCI_AIADDR).

AAC/MP4: Parsing issue

Some MP4 atoms were thought to have subatoms when they did not, causing too much data to be read. This made the file unplayable. The issue is now fixed.

IMA ADPCM encoding does not send encoded data

You can start encoding mode by setting the SM_ADPCM bit in SCI_MODE, then writing 0x300 to SCI_AIADDR.

When using this code, you should not load the patch described in the vs1053b datasheet.

MP3: Bit Reservoir Handling

MPEG Layer III provides a way to even out the bitrate consumption even with constant bitrate streams using the otherwise unused space of the previous frame, called bit reservoir. This allows the encoder to temporarily use more bits to encode difficult portions, for example transients.

However, when decoding is started in the middle of a stream and the decoder sees that this 'borrowed' space is used, it should also check if it actually contains valid data from the previous frames. Otherwise the first few frames may be decoded incorrectly, and cause disturbances to the sound.

The mp3 decoder version used in vs1053b does not calculate the amount of valid data in bit reservoir, so if you start decoding in a middle of an mp3 stream, decoding can access invalid data.

This patch aims to keep track of the data in bit reservoir and skip the decoding of mp3 frames that reference non-existing bit reservoir data. This will make the start of the decoding smooth.

1.2 New and Extended Features

FLAC Decoder

The patch also contains a FLAC decoder for lossless audio decompression. FLAC files can be played just like all other files by simply sending the file to SDI. See section 1.7.

Pause Mode

Play pause is traditionally implemented by stopping sending data to VS10xx. With low-bitrate songs, especially MIDI, decoding will continue for a long time because stream buffer is filled with data. This patch implements pause by stopping audio generation if bit 13 in SCI_MODE is set.

The pause mode can also be activated from parametric_x.reserved[2] bit 1, for compatibility with vs1063a.

Sample counter

This patch also adds a sample counter to help with streaming applications. See section 1.3.

Tags in SDI Data

This patch also allows you to interleave specific commands into the SDI data between mp3 frames or between files. See section 1.4.

VU Meter

SCI_STATUS bit 9 enables VU meter, which calculates a leaking peak sample value from both channels and returns the values in 1 dB resolution through SCI_AICTRL3. The high 8 bits represent the left channel, the low 8 bits the right channel.

Note that the following has changed from the previous release.

Values from 0 to 96 are valid for both channels. 0 means no sound, 96 is highest level. In practise the highest value will be 95, the signal is probably clipping if you get 96.

The calculation does not take the volume setting into account.

The VU meter takes about 0.6 MHz of processing power with 48 kHz samplerate.

Sample-Exact Rate Update

By default sample rate is updated whenever it changes. However, the audio buffer still contains samples with a different rate and those have not yet been played yet. The samples that are played with the wrong rate can sound like noise and are thus disturbing.

This patch synchronizes the samplerate change to occur at the right sample.

Sample Rate Fine Tuning

This patch also implements sample rate fine-tuning for special streaming applications. See section 1.5.

Mono / dual channel output mode

parametric_x.reserved[2] (address X:0x1e09) bit 0 selects mono output mode. In this mode the left and right channel data are mixed together. Volume control adjusts the result of this downmix (not the amount of left and right data in the result) for each output.

To activate the mono downmix mode, write 0x1e09 to SCI_WRAMADDR, then 0x0001 to SCI_WRAM.
To deactivate, write 0x1e09 to SCI_WRAMADDR, then 0x0000 to SCI_WRAM.

This bit works independently from the SM_DIFF bit in the mode register.

If parametric_x.reserved[2] (address X:0x1e09) bit 9 is also set, only one of the decoded channels is output from both channels. parametric_x.reserved[2] (address X:0x1e09) bit 10 selects whether the left channel (0) or right channel (1) is output.

To select only the left channel, write 0x1e09 to SCI_WRAMADDR, then 0x0201 to SCI_WRAM.

To select only the right channel, write 0x1e09 to SCI_WRAMADDR, then 0x0601 to SCI_WRAM.

See section 1.6 for the other bits in parametric_x.reserved[2] (playMode).

Note that the parametric structure is cleared after software reset, and when you restart the patch code (by writing 0x300 to AIADDR). When you perform either one, also remember to reactivate the mono mode.

LATM/LOAS Parser

This patch also implements a simplified LATM/LOAS parser for AAC. See section 1.8.

ID3v2 Parser

The ID3v2 tag can contain text data, but it can also contain image data or some other binary data. When the format identification routine sees data that matches one of the decodable audio formats, it calls the respective decoder. An inadvertent match may happen when the ID3 header contains binary data. If the respective decoder consumes too much data when trying to figure out how to play the data before giving up, the start of the actual audio data also gets eaten, and the file may not play at all.

The added ID3v2 parser detects the ID3 header, then skips the correct number of bytes, so that the format identification routine can start its work from the beginning of the actual audio data.

When ID3 header is detected, the value of HDAT1 changes to 0x4944. You can break out from the data skip loop with SM_OUT_OF_WAV, so the parser is compatible with the normal end-of-song processing.

Application Hook at 0x60

Normally zero samples are inserted to the audio FIFO whenever new data to be decoded does not arrive quickly enough and the audio buffer is getting too empty (less than 64 stereo samples).

However, vs1053 and vs1063 also have DAC FIFO underrun detection, which causes the audio output to behave cleanly even if new samples are not inserted. If there are no new samples to be sent to DAC, the signal will be faded slowly towards zero.

Writing 0x60 to SCI_AIADDR causes the zero sample insertion to be disabled. Additionally whenever any decoder outputs samples, value of 1 will be written to SCI_AICTRL0. The user must clear the value.

Write 0 to SCI_AIADDR to disable these features.

Note: this feature is not available with the version that has both FLAC and LATM support (vs1053b-patches-flac-latm).

1.3 Sample Counter

The 32-bit sample counter is designed to help when streaming Ogg Vorbis files. It tells the absolute number of the sample currently played through the DAC when Ogg Vorbis files are played back. With other file types the counter is free-running. The sample counter is located at the beginning of the X memory user area, at address 0x1800, and can be read through SCI.

The Ogg Vorbis Encoder application has a similar counter. This makes it possible to maintain synchronization between the encoder and the decoder with an accuracy of a few samples.

Note: The sample counter is valid only after at least the first 8 KiB of the Ogg Vorbis file has been played.

As the sample counter is 32 bits and the SCI interface is 16 bits, the most significant bits of the counter may change while it is being read. To prevent incorrect values, read the sample counter using the following, self-correcting code:

```
unsigned short ReadVS10xxRegister(unsigned short addr);
void WriteVS10xxRegister(unsigned short addr, unsigned short value);

unsigned long Read32BitsFromSCI(unsigned short memAddr) {
  unsigned short msbV1, lsb, msbV2;

  WriteVS10xxRegister(SCI_WRAMADDR, addr+1);
  msbV1 = (u_int16)ReadVS10xxRegister(SCI_WRAM);
  WriteVS10xxRegister(SCI_WRAMADDR, addr);
  lsb   = (u_int32)ReadVS10xxRegister(SCI_WRAM);
  msbV2 = (u_int16)ReadVS10xxRegister(SCI_WRAM);
  if (lsb < 0x8000U) {
    msbV1 = msbV2;
  }
  return ((u_int32)msbV1 << 16) | lsb;
}
```

The code is used like this:

```
unsigned long sampleCount = Read32BitsFromSCI(0x1800);
```

1.4 Tags in SDI Data

Tags are commands that can be written into the SDI data stream. The commands are queued with the data, and are executed when the tag is read by the decoder. The tags can only be inserted between MP3 frames or just before any other file format.

If you do not know where the MP3 frame boundary is, first send enough zeros to fill any partial frame. You can also check the current mp3 frame size from the SCI_HDAT0 and SCI_HDAT1 registers.

All commands are 6 bytes long. The first 4 bytes are “PlUg” (0x50 0x6c 0x55 0x67), the fifth byte is the command, the sixth byte contains the parameter. Currently supported tag commands are:

- `PlUgc<n>` - Cue point, copies parameter to AICTRL0.
- `PlUgv<n>` - Volume set, sets VOL to 0x0101*`n`.
- `PlUgt<n>` - Sample count, set to `n`.

Note that when you give sample count command 0..255, the value you see in sample counter is smaller than the set value until old samples have been played from the audio buffer.

1.5 Sample Rate Finetuning

Crystal and oscillator frequencies are never exact. They vary slightly depending on the accuracy of the physical crystal dimensions, and thus, the ambient temperature. When real-time audio data is transferred between systems, both ends have their own variances, and the difference must be adjusted for. This is why sample rate must be fine-tuned so that the receiving buffer neither gets empty nor overflows.

With VS10xx you can use the CLOCKF register to specify a higher clock than you are actually using to slow down the playback, or specify a lower clock to speed up the playback. However, in special applications the adjustment accuracy achieved this way may not be sufficient.

The adjustment accuracy of the DAC hardware is about 0.09 Hz. This patch allows you to fine-tune the sample rate in about $+-2$ ppm steps (for 48 kHz) from the parametric structure addresses 0x1e07 and 0x1e08 (parametric_x.reserved[0] and parametric_x.reserved[1]). Note: software reset and patches restart sets the value to zero.

The valid range of the fine tune depends on the XTAL. For 12.288 MHz XTALI the range is from -187000 to 511999.

```
unsigned short ReadVS10xxRegister(unsigned short addr);
void WriteVS10xxRegister(unsigned short addr, unsigned short value);

void AdjustRate(long ppm2) {
  WriteVS10xxRegister(SCI_WRAMADDR, 0x1e07);
  WriteVS10xxRegister(SCI_WRAM, ppm2);
  WriteVS10xxRegister(SCI_WRAM, ppm2 >> 16);

  /* oldClock4KHz = 0 forces adjustment calculation when rate checked. */
  WriteVS10xxRegister(SCI_WRAMADDR, 0x5b1c);
  WriteVS10xxRegister(SCI_WRAM, 0);

  /* Write to AUDATA or CLOCKF checks rate and recalculates adjustment. */
  WriteVS10xxRegister(SCI_AUDATA, ReadVS10xxRegister(SCI_AUDATA));
}
```

An example calculation for one semitone up or down:
$(1.0594631 - 1) * 512000 = 30445 =$ 0x76ed $\rightarrow$
write 0x76ed to 0x1e07 and 0 to 0x1e08 for one semitone up,
$(1/1.0594631 - 1) * 512000 = -28736 =$ 0xffff8fbf $\rightarrow$
write 0x8fbf to 0x1e07 and 0xffff to 0x1e08 for one semitone down.

When the samplerate is below 48000 Hz, the smallest adjustment steps do not change the rate. For example with 8000 Hz the adjustment value needs to change by 6 to affect the playback rate. However, adjustment value 1 can still result to different rate than value 0 due to rounding in the control value calculation.

Note: When XTALI is 12.288 MHz, the highest possible sample rate is 48000 Hz, so adjusting rate up for 48000 Hz files is not possible. If you need to adjust 48000 Hz audio, one solution is to use 13 MHz clock for 50781 Hz maximum rate. Another solution is to activate the 15/16 Resampler, see below.

1.6 15/16 Resampler

Note: the 15/16 Resampler can not be used while decoding a FLAC file!

If you use 12.288 MHz XTALI, the maximum possible samplerate is 48000 Hz. If you play audio from a real-time source, even when the source is an identical device, its rate can be a little lower or higher than our playback rate, due to differencies in crystals and ambient temperatures. If the source rate is higher than we can play, after a while the input buffer will overrun.

One way to resolve this is to use for example 13 MHz crystal so that you can play faster than 48 kHz. When you need to adjust the playback rate above 48000 Hz with 12.288 MHz crystal, you can use the 15/16 Resampler. This converts the audio to 45000 Hz, so you have plenty of room to increase the playback rate.

parametric_x.reserved[2] (address X:0x1e09) bit 7 enables the 15/16 Resampler and the appropriate compensation for the samplerate control value. To activate the down-sampler, write 0x1e09 to SCI_WRAMADDR, then 0x0080 to SCI_WRAM. The new sam-plerate tuning becomes active automatically.

To deactivate, write 0x1e09 to SCI_WRAMADDR, then 0x0000 to SCI_WRAM.

The resampler takes about 5 MHz of processing power.

parametric_x.reserved[2] (address X:0x1e09) bit 8 enables the Resampler without the samplerate compensation. With this you can test the Resampler. You should hear the pitch go up when you activate the Resampler with this bit.

parametric_x.playMode

parametric_x.reserved[2] is compatible with vs1063a `playMode`. With this patches package it provides mono and pause select bits. It also contains the new Resampler control bits.

Setting the pause bit will immediately stop audio sample output. Samples already in the audio buffer will be played, but stream buffer is not read until pause bit is cleared. The mono select averages left and right channel so LEFT and RIGHT outputs will be the same. Other bits are explained separately.

playMode		
bits	**Name**	**Usage**
10	PLAYMODE_DUAL_CHANNEL_SEL	Mono and Dual Channel set, 0=left,1=right
9	PLAYMODE_DUAL_CHANNEL	If also Mono mode set, 1=pick one channel
8	PLAYMODE_RESAMPLER_TEST	Resampler enable
7	PLAYMODE_RESAMPLER_ON	Resampler enable with rate compensation
1	PLAYMODE_PAUSE_ON	Pause enable
0	PLAYMODE_MONO_OUTPUT	Mono output select

1.7 FLAC Decoder

The patch also contains a FLAC decoder for lossless audio decompression. FLAC files can be played just like all other files by simply sending the file to SDI.

The FLAC decoder is not extensively tested, so if you find a file that you have problems with, please send it to us (support@vlsi.fi) for analysis.

Because of the high data rate, the requirements for data transfer are much higher than for lossy codecs. Because of compression, audio buffer being shorter than the default FLAC block size, and some design choices in the FLAC format itself, the peak data transfer rate must be even higher than the sustained data rate required for uncompressed WAV files.

The FLAC decoder lowers the peak data transfer requirement a little by providing a larger stream buffer (12 KiB).

Currently 16-bit FLAC files upto 48 kHz are tested to play smoothly in an actual system (SD card as storage). If you can manage high enough transfer rates you may be getting 24-bit files to play smoothly as well.

Files with more than 2 channels are matrixed to stereo, but note that these files will also take more processing power.

Both header CRC-8 and data CRC-16 are implemented in this version.

When FLAC format is detected, SCI_HDAT1 contains “fL” (0x664c).

You should send 12288 endFillBytes (0x55) instead of just 2050 when ending a file or jumping in the file.

Replay Gain is not yet read from Vorbis comments.

FLAC files also contain a metadata section. You should not jump in the file during these headers. DO_NOT_JUMP bit is now (since 1.95) correctly set during header decode.

Note: the multichannel matrixing feature and data CRC-16 are not available in the FLAC + LATM/LOAS version.

1.8 LATM/LOAS Parser

Optionally you can select a version that contains a simplified LATM/LOAS parser to support AAC-LC and HE-AAC in that container. Note that the parser is simplified and assumes only a single subframe, program and layer. Other configurations are considered as non-decodeable.

The data should be byte-aligned for the initial detection of the LATM/LOAS format (byte 0x56 followed by a value of 0xe0 through 0xff, which produces the 11-bit sync mark 0x2b7 in the top bits). You should also send at least two zero byte to SDI before the LATM/LOAS data to guarantee that the format is detected immediately from the first frame. If the data is not byte-aligned in this way, send 0x00 0x00 0x56 0xe0 to trigger the format detection. The decoder will then synchronize to the bitstream even if it is not byte-aligned.

SCI_HDAT1 contains 0x4c41 ('LA') when LATM/LOAS decoding is active. Decoding can be ended by the OUT_OF_WAV bit of the SCI_MODE register.

If no audio configuration is found, synchronization will time-out regardless of parametric_x.resync, and the decoder exits to the main decode loop and SCI_HDAT1 becomes 0. Once playback has started, the normal operation of parametric_x.resync is restored.

When changing streams, it is recommended to exit the LATM/LOAS decoding mode using OUT_OF_WAV and sending zero bytes until SCI_HDAT1 becomes 0.

The version with both FLAC and LATM does not include ID3v2 skipping.

If you come across a file that does not play correctly, please contact our support.

1.9 DSD64 Decoder

One version of the patch (vs1053b-patches-dsd) contains a DSD decoder for two channels with the 64x data rate. While this version is loaded, DSD files in .dff and .dsf containers can be played just like all other files by simply sending the file to SDI.

The DSD decoder is not extensively tested, so if you find a file that you have problems with, please send it to us (support@vlsi.fi) for analysis.

The data rate is double that of a 32-bit stereo WAV file, so the controller needs to handle data transfers efficiently. However, there is no peak rate requirement. Even the 4kB data blocks of the .dsf container are not an issue, because multiple result audio frames fit in the audio output FIFO at the same time. As long as the sustained data rate is high enough, you won't have an issue.

The DSD decoder only accepts stereo files and plays only 64x data rate files correctly.

When a valid DSD format is detected, SCI_HDAT1 contains "DS" (0x4444).

You should send 12288 endFillBytes (0x55) instead of just 2050 when ending a file or jumping in the file.

2 How to Load a Plugin

A plugin file (.plg) contains a data file that contains one unsigned 16-bit array called plugin. The file is in an interleaved and RLE compressed format. An example of a plugin array is:

```
const unsigned short plugin[10] = { /* Compressed plugin */
  0x0007, 0x0001, 0x8260,
  0x0006, 0x0002, 0x1234, 0x5678,
  0x0006, 0x8004, 0xabcd,
};
```

The vector is decoded as follows:

1. Read register address number `addr` and repeat number `n`.
2. If `(n & 0x8000U)`, write the next word `n` times to register `addr`.
3. Else write next `n` words to register `addr`.
4. Continue until array has been exhausted.

The example array first tells to write 0x8260 to register 7. Then write 2 words, 0x1234 and 0x5678, to register 6. Finally, write 0xabcd 4 times to register 6.

Assuming the array is in `plugin[]`, a full decoder in C language is provided below:

```
void WriteVS10xxRegister(unsigned short addr, unsigned short value);

void LoadUserCode(void) {
  int i = 0;

  while (i<sizeof(plugin)/sizeof(plugin[0])) {
    unsigned short addr, n, val;
    addr = plugin[i++];
    n = plugin[i++];
    if (n & 0x8000U) { /* RLE run, replicate n samples */
      n &= 0x7FFF;
      val = plugin[i++];
      while (n--) {
        WriteVS10xxRegister(addr, val);
      }
    } else {           /* Copy run, copy n samples */
      while (n--) {
        val = plugin[i++];
        WriteVS10xxRegister(addr, val);
      }
    }
  }
}
```

3 How to Use Old Loading Tables

Each patch contains two arrays: `atab` and `dtab`. `dtab` contains the data words to write, and `atab` gives the SCI registers to write the data values into. For example:

```
const unsigned char atab[] = { /* Register addresses */
    7, 6, 6, 6, 6
};
const unsigned short dtab[] = { /* Data to write */
    0x8260, 0x0030, 0x0717, 0xb080, 0x3c17
};
```

These arrays tell to write 0x8260 to SCI_WRAMADDR (register 7), then 0x0030, 0x0717, 0xb080, and 0x3c17 to SCI_WRAM (register 6). This sequence writes two 32-bit instruction words to instruction RAM starting from address 0x260. It is also possible to write 16-bit words to X and Y RAM. The following code loads the patch code into VS10xx memory.

```
/* A prototype for a function that writes to SCI */
void WriteVS10xxRegister(unsigned char sciReg, unsigned short data);

void LoadUserCode(void) {
  int i;
  for (i=0;i<sizeof(dtab)/sizeof(dtab[0]);i++) {
    WriteVS10xxRegister(atab[i]/*SCI register*/, dtab[i]/*data word*/);
  }
}
```

Patch code tables use mainly these two registers to apply patches, but they may also contain other SCI registers, especially SCI_AIADDR (10), which is the application code hook.

If different patch codes do not use overlapping memory areas, you can concatenate the data from separate patch arrays into one pair of `atab` and `dtab` arrays, and load them with a single `LoadUserCode()`.