

VS1053B PATCHES AND FLAC DECODER

VSMPG “VLSI Solution Audio Decoder”

Project Code:
Project Name: VSMPG

Revision History			
Rev.	Date	Author	Description
1.3	2010-06-01	PO	Fixed 't' and 'v' tags for the non-FLAC version.
1.2	2010-04-22	PO	Bit 13 of SCLMODE becomes pause bit, bit 8 of SCLSTATUS is copied to bit 0 to allow 1.65V reference voltage. SS_DO_NOT_JUMP cleared at startup. Tags. New AAC clock change function. VU meter.
1.1	2009-06-23	PO	All VS1053b patches combined with optional FLAC decoder.
1.0	2009-01-19	HH	Playback sample counter can now be read.
0.6	2008-04-28	PO	Ogg streams with the high bit set in the serial number did not play. Serial number is now ignored.
0.5	2007-11-30	PO	Initial version.

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).
- 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 cleared at software reset.

In addition the first trial version of a FLAC decoder is available in the other version.

When the patch is loaded, it starts automatically (writes 0x0300 to AIADDR) and restarts the system (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.

Chip	File	IRAM	Description
VS1053B	vs1053b-patches.c	0x50..0x14a, 0x300 .. 0x419	old array format
VS1053B	vs1053b-patches.plg	0x50..0x14a, 0x300 .. 0x419	compressed plugin
VS1053B	vs1053b-patches-flac.c	0x50..0xbda	with FLAC decoding
VS1053B	vs1053b-patches-flac.plg	0x50..0xbda	with FLAC decoding

This patch uses the application address to start automatically (the last entry in the patch tables writes to SCI_AIADDR), but does not use it afterwards.

There are two versions: one with all of the patches, and another with all patches and the FLAC decoder.

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.

This patch is **not** compatible with 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.

1.1 Fixes in Detail

Vorbis: Occasional windowing overflow

Occasional windowing overflow which could cause audible clicks is eliminated.

You can use the included 8-second sample `BlueHotelShort.ogg` to test that the patch has been correctly loaded. Without the patch you can hear a clear snap in the right channel at about halfway into the sample. With the patch the sample plays without the disturbance.

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.

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 periodically without the routine dropping decoding features. This causes small clicks on the audio.

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.

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, but they seem to be insufficient.

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.

SCLSTATUS 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 (SCLSTATUS becomes 0x0040), then write 0x0101 to SCLSTATUS.

SS_DO_NOT_JUMP

The vs1053b firmware does not clear the DO_NOT_JUMP bit from SCLSTATUS during software reset, so sometimes fast forward and rewind appear to be forbidden unnecessarily. This patch clears the bit at startup.

IMA ADPCM encoding does not send encoded data

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

When using this code, you do not need to load the patch described in the vs1053b datasheet.

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.

Sample counter

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

Tags in SDI Data

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

VU Meter

SCI.STATUS bit 9 enables VU meter, which calculates the peak sample value from both channels and returns the values in 3 dB resolution through SCI.AICTRL3. The high 8 bits represent the left channel, the low 8 bits the right channel.

Values from 0 to 31 are valid for both channels.

The VU meter takes about 0.2 MHz of processing power with 48 kHz samplerate.

1.2 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 8KiB 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(port, SCI_WRAMADDR, addr+1);
    msbV1 = (u_int16)ReadVS10xxRegister(port, SCI_WRAM);
    WriteVS10xxRegister(port, SCI_WRAMADDR, addr);
    lsb = (u_int32)ReadVS10xxRegister(port, SCI_WRAM);
    msbV2 = (u_int16)ReadVS10xxRegister(port, SCI_WRAM);
    if (lsb < 0x8000U) {
        msbV1 = msbV2;
    }
    return ((u_int32)msbV1 << 16) | lsb;
}
```

The code is used like this:

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

1.3 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 "PIUg" (0x50 0x6c 0x55 0x67), the fifth byte is the command, the sixth byte contains the parameter. Currently supported tag commands are:

- PIUgc<n> - Cue point, copies parameter to AICTRL0.
- PIUgv<n> - Volume set, sets VOL to 0x0101*n.
- PIUgs<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.

1.4 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.

Note: This is the first test version of the FLAC decoder. It 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.

You should send 12288 endFillBytes instead of just 2050 when ending a file.

Replay Gain is not yet read from Vorbis comments.

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);
            }
        }
        i++;
    }
}
```