

VS1053b - Ogg Vorbis/MP3/AAC/WMA/FLAC/ MIDI AUDIO CODEC CIRCUIT

Features

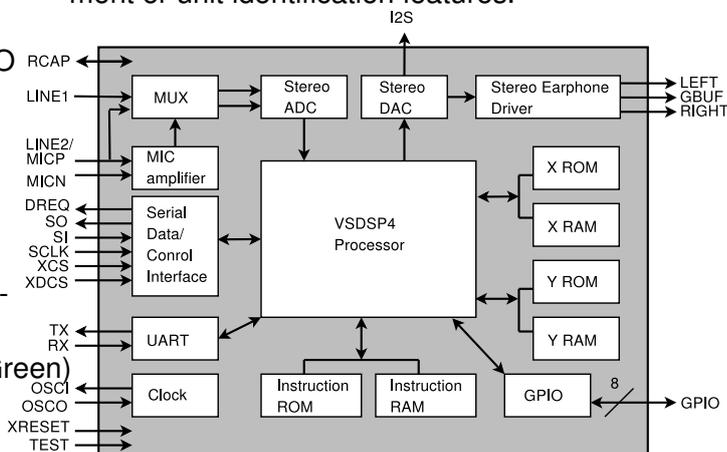
- Decodes
Ogg Vorbis;
MP3 = MPEG 1 & 2 audio layer III (CBR +VBR +ABR);
MP1/MP2 = layers I & II optional;
MPEG4/2 AAC-LC(+PNS),
HE-AAC v2 (Level 3) (SBR + PS);
WMA 4.0/4.1/7/8/9 all profiles (5-384 kbps);
General MIDI 1 / SP-MIDI format 0 files;
FLAC with software plugin;
WAV (PCM + IMA ADPCM)
- Encodes Ogg Vorbis w/ software plugin
- Encodes stereo IMA ADPCM / PCM
- Streaming support for MP3 and WAV
- EarSpeaker Spatial Processing
- Bass and treble controls
- Operates with a single 12..13 MHz clock
- Can also be used with a 24..26 MHz clock
- Internal PLL clock multiplier
- Low-power operation
- High-quality on-chip stereo DAC with no phase error between channels
- Zero-cross detection for smooth volume change
- Stereo earphone driver capable of driving a 30 Ω load
- Quiet power-on and power-off
- I2S interface for external DAC
- Separate voltages for analog, digital, I/O
- On-chip RAM for user code and data
- Serial control and data interfaces
- Can be used as a slave co-processor
- SPI flash boot for special applications
- UART for debugging purposes
- New functions may be added with software and upto 8 GPIO pins
- Lead-free RoHS-compliant package (Green)

Description

VS1053b is an Ogg Vorbis/MP3/AAC/WMA/FLAC/WAVMIDI audio decoder as well as an PCM/IMA ADPCM/Ogg Vorbis encoder on a single chip. It contains a high-performance, proprietary low-power DSP processor core VS_DSP⁴, data memory, 16 KiB instruction RAM and 0.5+ KiB data RAM for user applications running simultaneously with any built-in decoder, serial control and input data interfaces, upto 8 general purpose I/O pins, an UART, as well as a high-quality variable-sample-rate stereo ADC (mic, line, line + mic or 2×line) and stereo DAC, followed by an earphone amplifier and a common voltage buffer.

VS1053b receives its input bitstream through a serial input bus, which it listens to as a system slave. The input stream is decoded and passed through a digital volume control to an 18-bit oversampling, multi-bit, sigma-delta DAC. The decoding is controlled via a serial control bus. In addition to the basic decoding, it is possible to add application specific features, like DSP effects, to the user RAM memory.

Optional factory-programmable unique chip ID provides basis for digital rights management or unit identification features.



Contents

| | |
|--|-----------|
| VS1053 | 1 |
| Table of Contents | 2 |
| List of Figures | 5 |
| 1 Licenses | 6 |
| 2 Disclaimer | 6 |
| 3 Definitions | 6 |
| 4 Characteristics & Specifications | 7 |
| 4.1 Absolute Maximum Ratings | 7 |
| 4.2 Recommended Operating Conditions | 7 |
| 4.3 Analog Characteristics | 8 |
| 4.4 Power Consumption | 9 |
| 4.5 Digital Characteristics | 9 |
| 4.6 Switching Characteristics - Boot Initialization | 9 |
| 5 Packages and Pin Descriptions | 10 |
| 5.1 Packages | 10 |
| 5.1.1 LQFP-48 | 10 |
| 6 Connection Diagram, LQFP-48 | 13 |
| 7 SPI Buses | 15 |
| 7.1 SPI Bus Pin Descriptions | 15 |
| 7.1.1 VS10xx Native Modes (New Mode, recommended) | 15 |
| 7.1.2 VS1001 Compatibility Mode (deprecated, do not use in new designs) | 15 |
| 7.2 Data Request Pin DREQ | 16 |
| 7.3 Serial Protocol for Serial Data Interface (SPI / SDI) | 17 |
| 7.3.1 SDI in VS10xx Native Modes (New Mode, recommended) | 17 |
| 7.3.2 SDI Timing Diagram in VS10xx Native Modes (New Mode) | 18 |
| 7.3.3 SDI in VS1001 Compatibility Mode (deprecated, do not use in new designs) | 19 |
| 7.3.4 Passive SDI Mode (deprecated, do not use in new designs) | 19 |
| 7.4 Serial Protocol for Serial Command Interface (SPI / SCI) | 20 |
| 7.4.1 SCI Read | 20 |
| 7.4.2 SCI Write | 21 |
| 7.4.3 SCI Multiple Write | 21 |
| 7.4.4 SCI Timing Diagram | 22 |
| 7.5 SPI Examples with SM_SDINew and SM_SDISHARED set | 23 |
| 7.5.1 Two SCI Writes | 23 |
| 7.5.2 Two SDI Bytes | 23 |
| 7.5.3 SCI Operation in Middle of Two SDI Bytes | 24 |
| 8 Supported Audio Decoder Formats | 25 |
| 8.1 Supported MP3 (MPEG layer III) Formats | 25 |
| 8.2 Supported MP2 (MPEG layer II) Formats | 26 |

| | | |
|-----------|---|-----------|
| 8.3 | Supported MP1 (MPEG layer I) Formats | 26 |
| 8.4 | Supported Ogg Vorbis Formats | 26 |
| 8.5 | Supported AAC (ISO/IEC 13818-7 and ISO/IEC 14496-3) Formats | 27 |
| 8.6 | Supported WMA Formats | 29 |
| 8.7 | Supported FLAC Formats | 30 |
| 8.8 | Supported RIFF WAV Formats | 30 |
| 8.9 | Supported MIDI Formats | 31 |
| 9 | Functional Description | 33 |
| 9.1 | Main Features | 33 |
| 9.2 | Data Flow of VS1053b | 34 |
| 9.3 | EarSpeaker Spatial Processing | 35 |
| 9.4 | Serial Data Interface (SDI) | 36 |
| 9.5 | Serial Control Interface (SCI) | 36 |
| 9.6 | SCI Registers | 37 |
| 9.6.1 | SCI_MODE (RW) | 38 |
| 9.6.2 | SCI_STATUS (RW) | 40 |
| 9.6.3 | SCI_BASS (RW) | 41 |
| 9.6.4 | SCI_CLOCKF (RW) | 42 |
| 9.6.5 | SCI_DECODE_TIME (RW) | 43 |
| 9.6.6 | SCI_AUDATA (RW) | 43 |
| 9.6.7 | SCI_WRAM (RW) | 43 |
| 9.6.8 | SCI_WRAMADDR (W) | 43 |
| 9.6.9 | SCI_HDAT0 and SCI_HDAT1 (R) | 44 |
| 9.6.10 | SCI_AIADDR (RW) | 46 |
| 9.6.11 | SCI_VOL (RW) | 47 |
| 9.6.12 | SCI_AICTRL[x] (RW) | 47 |
| 10 | Operation | 48 |
| 10.1 | Clocking | 48 |
| 10.2 | Hardware Reset | 48 |
| 10.3 | Software Reset | 48 |
| 10.4 | Low Power Mode | 49 |
| 10.5 | Play and Decode | 49 |
| 10.5.1 | Playing a Whole File | 50 |
| 10.5.2 | Cancelling Playback | 50 |
| 10.5.3 | Fast Play | 50 |
| 10.5.4 | Fast Forward and Rewind without Audio | 51 |
| 10.5.5 | Maintaining Correct Decode Time | 51 |
| 10.6 | Feeding PCM data | 52 |
| 10.7 | Ogg Vorbis Recording | 52 |
| 10.8 | PCM/ADPCM Recording | 53 |
| 10.8.1 | Activating ADPCM Mode | 53 |
| 10.8.2 | Reading PCM / IMA ADPCM Data | 54 |
| 10.8.3 | Adding a PCM RIFF Header | 55 |
| 10.8.4 | Adding an IMA ADPCM RIFF Header | 56 |
| 10.8.5 | Playing ADPCM Data | 57 |
| 10.8.6 | Sample Rate Considerations | 57 |
| 10.8.7 | Record Monitoring Volume | 57 |
| 10.9 | SPI Boot | 59 |
| 10.10 | Real-Time MIDI | 59 |

| | |
|---|-----------|
| 10.11 Extra Parameters | 60 |
| 10.11.1 Common Parameters | 61 |
| 10.11.2 WMA | 62 |
| 10.11.3 AAC | 63 |
| 10.11.4 Midi | 64 |
| 10.11.5 Ogg Vorbis | 64 |
| 10.12 SDI Tests | 66 |
| 10.12.1 Sine Test | 66 |
| 10.12.2 Pin Test | 66 |
| 10.12.3 SCI Test | 66 |
| 10.12.4 Memory Test | 67 |
| 10.12.5 New Sine and Sweep Tests | 67 |
| 11 VS1053b Registers | 69 |
| 11.1 Who Needs to Read This Chapter | 69 |
| 11.2 The Processor Core | 69 |
| 11.3 VS1053b Hardware DAC Audio Paths | 70 |
| 11.4 VS1053b Hardware ADC Audio Paths | 71 |
| 11.5 VS1053b Memory Map | 72 |
| 11.6 SCI Hardware Registers | 72 |
| 11.7 Serial Data Interface (SDI) Registers | 72 |
| 11.8 DAC Registers | 73 |
| 11.9 PLL Controller | 73 |
| 11.10 GPIO | 75 |
| 11.11 Interrupt Control | 76 |
| 11.12 UART | 77 |
| 11.12.1 UART Registers | 77 |
| 11.12.2 Status UART_STATUS | 77 |
| 11.12.3 Data UART_DATA | 78 |
| 11.12.4 Data High UART_DATAH | 78 |
| 11.12.5 Divider UART_DIV | 78 |
| 11.12.6 UART Interrupts and Operation | 79 |
| 11.13 Timers | 80 |
| 11.13.1 Timer Registers | 80 |
| 11.13.2 Configuration TIMER_CONFIG | 80 |
| 11.13.3 Configuration TIMER_ENABLE | 81 |
| 11.13.4 Timer X Startvalue TIMER_Tx[L/H] | 81 |
| 11.13.5 Timer X Counter TIMER_TxCNT[L/H] | 81 |
| 11.13.6 Timer Interrupts | 81 |
| 11.14 I2S DAC Interface | 82 |
| 11.15 Analog-to-Digital Converter (ADC) | 83 |
| 11.16 Resampler SampleRate Converter (SRC) | 84 |
| 11.17 Sidestream Sigma-Delta Modulator (SDM) | 85 |
| 12 Version Changes | 86 |
| 12.1 Changes Between VS1033c and VS1053a/b Firmware, 2007-03-08 | 86 |
| 13 Document Version Changes | 88 |
| 14 Contact Information | 89 |

List of Figures

| | | |
|----|--|----|
| 1 | Pin configuration, LQFP-48. | 10 |
| 2 | VS1053b in LQFP-48 packaging. | 10 |
| 3 | Typical connection diagram using LQFP-48. | 13 |
| 4 | SDI in VS10xx Native Mode, single-byte transfer | 17 |
| 5 | SDI in VS10xx Native Mode, multi-byte transfer, $X \geq 1$ | 17 |
| 6 | SDI timing diagram | 18 |
| 7 | SDI in VS1001 Mode - one byte transfer. Do not use in new designs! | 19 |
| 8 | SDI in VS1001 Mode - two byte transfer. Do not use in new designs! | 19 |
| 9 | SCI word read | 20 |
| 10 | SCI word write | 21 |
| 11 | SCI multiple word write | 21 |
| 12 | SPI timing diagram | 22 |
| 13 | Two SCI operations | 23 |
| 14 | Two SDI bytes | 23 |
| 15 | Two SDI bytes separated by an SCI operation | 24 |
| 16 | Data flow of VS1053b. | 34 |
| 17 | EarSpeaker externalized sound sources vs. normal inside-the-head sound | 35 |
| 18 | VS1053b ADC and DAC data paths with some data registers | 70 |
| 19 | VS1053b ADC and DAC data paths with some data registers | 71 |
| 20 | RS232 serial interface protocol | 77 |
| 21 | I2S interface, 192 kHz. | 82 |

1 Licenses

MPEG Layer-3 audio decoding technology licensed from Fraunhofer IIS and Thomson.

Note: If you enable Layer I and Layer II decoding, you are liable for any patent issues that may arise from using these formats. Joint licensing of MPEG 1.0 / 2.0 Layer III does not cover all patents pertaining to layers I and II.

VS1053b contains WMA decoding technology from Microsoft.

This product is protected by certain intellectual property rights of Microsoft and cannot be used or further distributed without a license from Microsoft.

VS1053b contains AAC technology (ISO/IEC 13818-7 and ISO/IEC 14496-3) which cannot be used without a proper license from Via Licensing Corporation or individual patent holders.

VS1053b contains spectral band replication (SBR) and parametric stereo (PS) technologies developed by Coding Technologies. Licensing of SBR is handled within MPEG4 through Via Licensing Corporation. Licensing of PS is handled with Coding Technologies.

See <http://www.codingtechnologies.com/licensing/aacplus.htm> for more information.

To the best of our knowledge, if the end product does not play a specific format that otherwise would require a customer license: MPEG 1.0/2.0 layers I and II, WMA, or AAC, the respective license should not be required. Decoding of MPEG layers I and II are disabled by default, and WMA and AAC format exclusion can be easily performed based on the contents of the SCI_HDAT1 register. Also PS and SBR decoding can be separately disabled.

2 Disclaimer

All properties and figures are subject to change.

3 Definitions

B Byte, 8 bits.

b Bit.

Ki “Kibi” = 2^{10} = 1024 (IEC 60027-2).

Mi “Mebi” = 2^{20} = 1048576 (IEC 60027-2).

VS_DSP VLSI Solution’s DSP core.

W Word. In VS_DSP, instruction words are 32-bit and data words are 16-bit wide.

4 Characteristics & Specifications

4.1 Absolute Maximum Ratings

| Parameter | Symbol | Min | Max | Unit |
|---|--------|------|------------------------|------|
| Analog Positive Supply | AVDD | -0.3 | 3.6 | V |
| Digital Positive Supply | CVDD | -0.3 | 1.85 | V |
| I/O Positive Supply | IOVDD | -0.3 | 3.6 | V |
| Current at Any Non-Power Pin ¹ | | | ±50 | mA |
| Voltage at Any Digital Input | | -0.3 | IOVDD+0.3 ² | V |
| Operating Temperature | | -30 | +85 | °C |
| Storage Temperature | | -65 | +150 | °C |

¹ Higher current can cause latch-up.

² Must not exceed 3.6 V

4.2 Recommended Operating Conditions

| Parameter | Symbol | Min | Typ | Max | Unit |
|---|-----------|------|--------|------|------|
| Ambient Operating Temperature | | -30 | | +85 | °C |
| Analog and Digital Ground ¹ | AGND DGND | | 0.0 | | V |
| Positive Analog, REF=1.23V | AVDD | 2.5 | 2.8 | 3.6 | V |
| Positive Analog, REF=1.65V ² | AVDD | 3.3 | 3.3 | 3.6 | V |
| Positive Digital | CVDD | 1.7 | 1.8 | 1.85 | V |
| I/O Voltage | IOVDD | 1.8 | 2.8 | 3.6 | V |
| Input Clock Frequency ³ | XTALI | 12 | 12.288 | 13 | MHz |
| Internal Clock Frequency | CLKI | 12 | 36.864 | 55.3 | MHz |
| Internal Clock Multiplier ⁴ | | 1.0× | 3.0× | 4.5× | |
| Master Clock Duty Cycle | | 40 | 50 | 60 | % |

¹ Must be connected together as close the device as possible for latch-up immunity.

² Reference voltage can be internally selected between 1.23V and 1.65V, see section 9.6.2.

³ The maximum sample rate that can be played with correct speed is XTALI/256 (or XTALI/512 if SM_CLK_RANGE is set). Thus, XTALI must be at least 12.288 MHz (24.576 MHz) to be able to play 48 kHz at correct speed.

⁴ Reset value is 1.0×. Recommended SC_MULT=3.5×, SC_ADD=1.0× (SCI_CLOCKF=0x8800). Do not exceed maximum specification for CLKI.

4.3 Analog Characteristics

Unless otherwise noted: AVDD=3.3V, CVDD=1.8V, IOVDD=2.8V, REF=1.65V, TA=-30...+85°C, XTALI=12..13MHz, Internal Clock Multiplier 3.5×. DAC tested with 1307.894 Hz full-scale output sinewave, measurement bandwidth 20..20000 Hz, analog output load: LEFT to GBUF 30 Ω, RIGHT to GBUF 30 Ω. Microphone test amplitude 48 mVpp, $f_s=1$ kHz, Line input test amplitude 1.26 V, $f_s=1$ kHz.

| Parameter | Symbol | Min | Typ | Max | Unit |
|--|--------|------|-------------------|-------------------|---------|
| DAC Resolution | | | 18 | | bits |
| Total Harmonic Distortion | THD | | | 0.07 | % |
| Third Harmonic Distortion | | | | 0.02 | % |
| Dynamic Range (DAC unmuted, A-weighted) | IDR | | 100 | | dB |
| S/N Ratio (full scale signal) | SNR | | 94 | | dB |
| Interchannel Isolation (Cross Talk), 600Ω + GBUF | | | 80 | | dB |
| Interchannel Isolation (Cross Talk), 30Ω + GBUF | | | 53 | | dB |
| Interchannel Gain Mismatch | | -0.5 | | 0.5 | dB |
| Frequency Response | | -0.1 | | 0.1 | dB |
| Full Scale Output Voltage (Peak-to-peak) | | 1.64 | 1.85 ¹ | 2.06 | Vpp |
| Deviation from Linear Phase | | | | 5 | ° |
| Analog Output Load Resistance | AOLR | 16 | 30 ² | | Ω |
| Analog Output Load Capacitance | | | | 100 | pF |
| Microphone input amplifier gain | MICG | | 26 | | dB |
| Microphone input amplitude | | | 48 | 140 ³ | mVpp AC |
| Microphone Total Harmonic Distortion | MTHD | | 0.03 | 0.07 | % |
| Microphone S/N Ratio | MSNR | 60 | 70 | | dB |
| Microphone input impedances, per pin | | | 45 | | kΩ |
| Line input amplitude | | | 2500 | 2800 ³ | mVpp AC |
| Line input Total Harmonic Distortion | LTHD | | 0.005 | 0.014 | % |
| Line input S/N Ratio | LSNR | 85 | 90 | | dB |
| Line input impedance | | | 80 | | kΩ |

¹ 3.0 volts can be achieved with +-to-+ wiring for mono difference sound.

² AOLR may be much lower, but below *Typical* distortion performance may be compromised.

³ Above typical amplitude the Harmonic Distortion increases.

4.4 Power Consumption

Tested with an Ogg Vorbis 128 kbps sample and generated sine. Output at full volume. Internal clock multiplier 3.0×. TA=+25°C.

| Parameter | Min | Typ | Max | Unit |
|---|-----|------|------|------|
| Power Supply Consumption AVDD, Reset | | 0.6 | 5.0 | μA |
| Power Supply Consumption CVDD = 1.8V, Reset | | 12 | 20.0 | μA |
| Power Supply Consumption AVDD, sine test, 30 Ω + GBUF | 30 | 36.9 | 60 | mA |
| Power Supply Consumption CVDD = 1.8V, sine test | 8 | 10 | 15 | mA |
| Power Supply Consumption AVDD, no load | | 5 | | mA |
| Power Supply Consumption AVDD, output load 30 Ω | | 11 | | mA |
| Power Supply Consumption AVDD, 30 Ω + GBUF | | 11 | | mA |
| Power Supply Consumption CVDD = 1.8V | | 11 | | mA |

4.5 Digital Characteristics

| Parameter | Min | Max | Unit |
|---|-----------|------------------------|------|
| High-Level Input Voltage (xRESET, XTALI, XTALO) | 0.7×IOVDD | IOVDD+0.3 ¹ | V |
| High-Level Input Voltage (other input pins) | 0.7×CVDD | IOVDD+0.3 ¹ | V |
| Low-Level Input Voltage | -0.2 | 0.3×CVDD | V |
| High-Level Output Voltage at XTALO = -0.1 mA | 0.7×IOVDD | | V |
| Low-Level Output Voltage at XTALO = 0.1 mA | | 0.3×IOVDD | V |
| High-Level Output Voltage at I _O = -1.0 mA | 0.7×IOVDD | | V |
| Low-Level Output Voltage at I _O = 1.0 mA | | 0.3×IOVDD | V |
| Input Leakage Current | -1.0 | 1.0 | μA |
| SPI Input Clock Frequency ² | | $\frac{CLKI}{7}$ | MHz |
| Rise time of all output pins, load = 50 pF | | 50 | ns |

¹ Must not exceed 3.6V

² Value for SCI reads. SCI and SDI writes allow $\frac{CLKI}{4}$.

4.6 Switching Characteristics - Boot Initialization

| Parameter | Symbol | Min | Max | Unit |
|-----------------------------------|--------|-------|--------------------|-------|
| XRESET active time | | 2 | | XTALI |
| XRESET inactive to software ready | | 22000 | 50000 ¹ | XTALI |
| Power on reset, rise time to CVDD | | 10 | | V/s |

¹ DREQ rises when initialization is complete. You should not send any data or commands before that.

5 Packages and Pin Descriptions

5.1 Packages

LPQFP-48 is a lead (Pb) free and also RoHS compliant package. RoHS is a short name of *Directive 2002/95/EC on the restriction of the use of certain hazardous substances in electrical and electronic equipment.*

5.1.1 LQFP-48

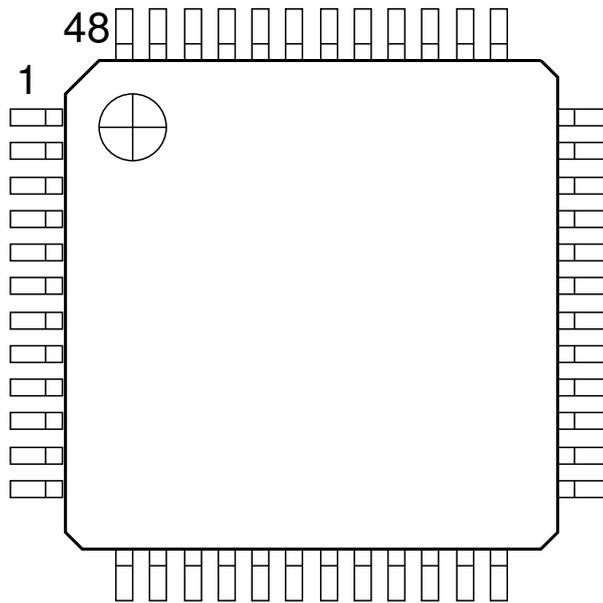


Figure 1: Pin configuration, LQFP-48.

LQFP-48 package dimensions are at <http://www.vlsi.fi/>.

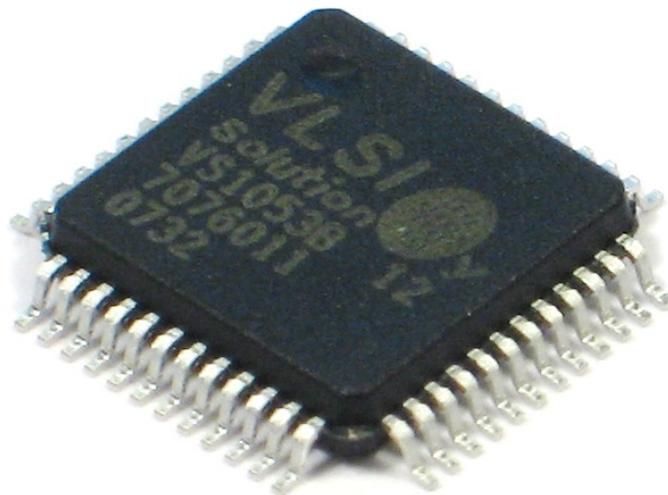


Figure 2: VS1053b in LQFP-48 packaging.

| Pad Name | LQFP Pin | Pin Type | Function |
|--------------------------------|----------|----------|---|
| MICP / LINE1 | 1 | AI | Positive differential mic input, self-biasing / Line-in 1 |
| MICN | 2 | AI | Negative differential mic input, self-biasing |
| XRESET | 3 | DI | Active low asynchronous reset, schmitt-trigger input |
| DGND0 | 4 | DGND | Core & I/O ground |
| CVDD0 | 5 | CPWR | Core power supply |
| IOVDD0 | 6 | IOPWR | I/O power supply |
| CVDD1 | 7 | CPWR | Core power supply |
| DREQ | 8 | DO | Data request, input bus |
| GPIO2 / DCLK ¹ | 9 | DIO | General purpose IO 2 / serial input data bus clock |
| GPIO3 / SDATA ¹ | 10 | DIO | General purpose IO 3 / serial data input |
| GPIO6 / I2S_SCLK ³ | 11 | DIO | General purpose IO 6 / I2S_SCLK |
| GPIO7 / I2S_SDATA ³ | 12 | DIO | General purpose IO 7 / I2S_SDATA |
| XDCS / BSYNC ¹ | 13 | DI | Data chip select / byte sync |
| IOVDD1 | 14 | IOPWR | I/O power supply |
| VCO | 15 | DO | For testing only (Clock VCO output) |
| DGND1 | 16 | DGND | Core & I/O ground |
| XTALO | 17 | AO | Crystal output |
| XTALI | 18 | AI | Crystal input |
| IOVDD2 | 19 | IOPWR | I/O power supply |
| DGND2 | 20 | DGND | Core & I/O ground |
| DGND3 | 21 | DGND | Core & I/O ground |
| DGND4 | 22 | DGND | Core & I/O ground |
| XCS | 23 | DI | Chip select input (active low) |
| CVDD2 | 24 | CPWR | Core power supply |
| GPIO5 / I2S_MCLK ³ | 25 | DIO | General purpose IO 5 / I2S_MCLK |
| RX | 26 | DI | UART receive, connect to IOVDD if not used |
| TX | 27 | DO | UART transmit |
| SCLK | 28 | DI | Clock for serial bus |
| SI | 29 | DI | Serial input |
| SO | 30 | DO3 | Serial output |
| CVDD3 | 31 | CPWR | Core power supply |
| XTEST | 32 | DI | Reserved for test, connect to IOVDD |
| GPIO0 | 33 | DIO | Gen. purp. IO 0 (SPIBOOT), use 100 k Ω pull-down resistor ² |
| GPIO1 | 34 | DIO | General purpose IO 1 |
| GND | 35 | DGND | I/O Ground |
| GPIO4 / I2S_LROUT ³ | 36 | DIO | General purpose IO 4 / I2S_LROUT |
| AGND0 | 37 | APWR | Analog ground, low-noise reference |
| AVDD0 | 38 | APWR | Analog power supply |
| RIGHT | 39 | AO | Right channel output |
| AGND1 | 40 | APWR | Analog ground |
| AGND2 | 41 | APWR | Analog ground |
| GBUF | 42 | AO | Common buffer for headphones, do NOT connect to ground! |
| AVDD1 | 43 | APWR | Analog power supply |
| RCAP | 44 | AIO | Filtering capacitance for reference |
| AVDD2 | 45 | APWR | Analog power supply |
| LEFT | 46 | AO | Left channel output |
| AGND3 | 47 | APWR | Analog ground |
| LINE2 | 48 | AI | Line-in 2 (right channel) |

¹ First pin function is active in New Mode, latter in Compatibility Mode.

² Unless pull-down resistor is used, SPI Boot is tried. See Chapter 10.9 for details.

³ If I2S_CF_ENA is '0' the pins are used for GPIO. See Chapter 11.14 for details.

Pin types:

| Type | Description |
|------|--|
| DI | Digital input, CMOS Input Pad |
| DO | Digital output, CMOS Input Pad |
| DIO | Digital input/output |
| DO3 | Digital output, CMOS Tri-stated Output Pad |
| AI | Analog input |

| Type | Description |
|-------|-------------------------|
| AO | Analog output |
| AIO | Analog input/output |
| APWR | Analog power supply pin |
| DGND | Core or I/O ground pin |
| CPWR | Core power supply pin |
| IOPWR | I/O power supply pin |

6 Connection Diagram, LQFP-48

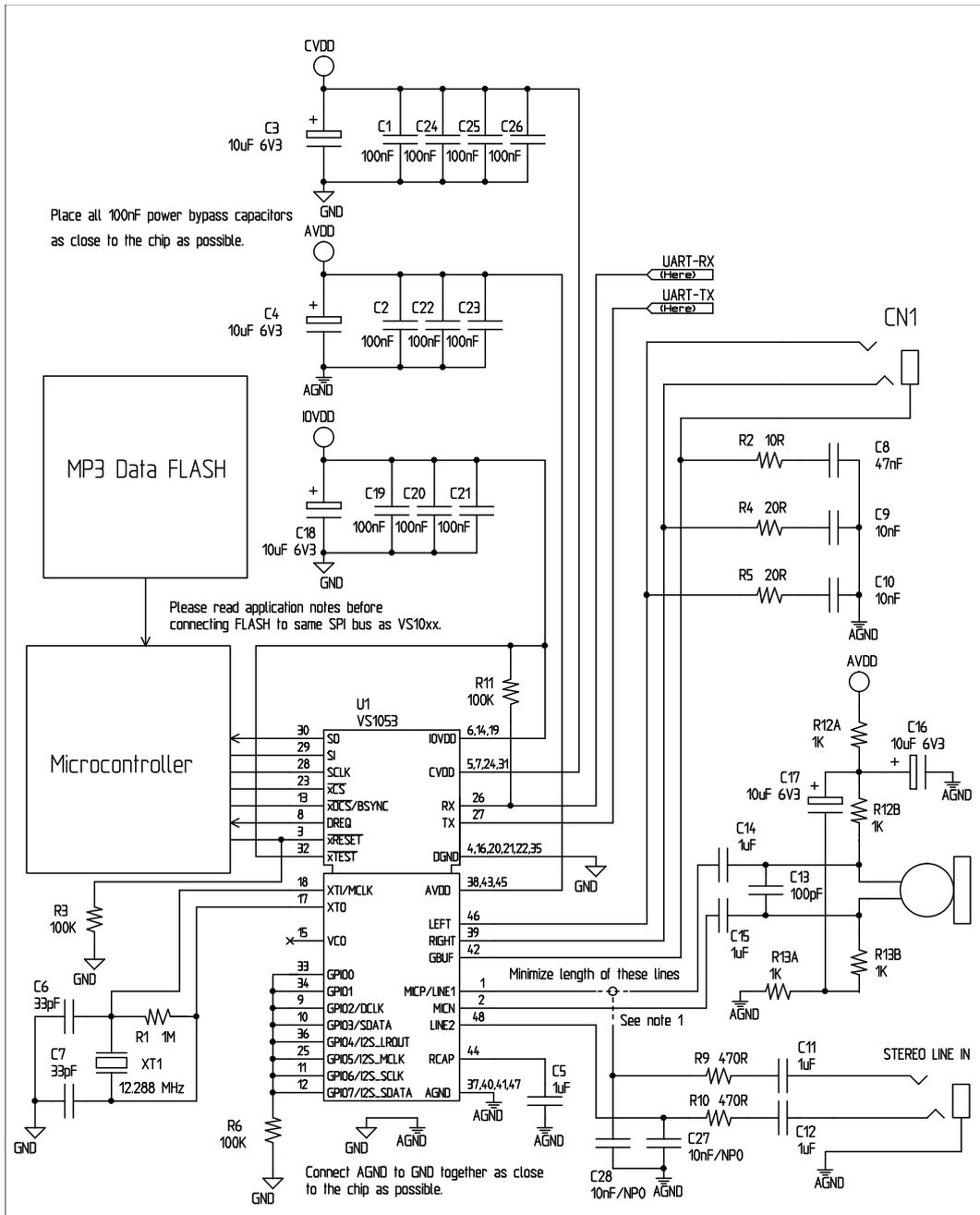


Figure 3: Typical connection diagram using LQFP-48.

Figure 3 shows a typical connection diagram for VS1053.

Figure Note 1: Connect either Microphone In or Line In, but not both at the same time.

Note: This connection assumes SM_SDINew is active (see Chapter 9.6.1). If also SM_SDISHARE is used, xDCS should be tied low or high (see Chapter 7.1.1).

The common buffer GBUF can be used for common voltage (1.23 V) for earphones. This will eliminate the need for large isolation capacitors on line outputs, and thus the audio output pins from VS1053b may be connected directly to the earphone connector.

GBUF must NOT be connected to ground under any circumstances. If GBUF is not used, LEFT and RIGHT must be provided with coupling capacitors. To keep GBUF stable, you should always have the resistor and capacitor even when GBUF is not used. See application notes for details.

Unused GPIO pins should have a pull-down resistor. Unused line and microphone inputs should not be connected.

If UART is not used, RX should be connected to IOVDD and TX be unconnected.

Do not connect any external load to XTALO.

7 SPI Buses

The SPI Bus - which was originally used in some Motorola devices - has been used for both VS1053b's Serial Data Interface SDI (Chapters 7.3 and 9.4) and Serial Control Interface SCI (Chapters 7.4 and 9.5).

7.1 SPI Bus Pin Descriptions

7.1.1 VS10xx Native Modes (New Mode, recommended)

These modes are active on VS1053b when SM_SDINew is set to 1 (default at startup). DCLK and SDATA are not used for data transfer and they can be used as general-purpose I/O pins (GPIO2 and GPIO3). BSYNC function changes to data interface chip select (XDCS).

| SDI Pin | SCI Pin | Description |
|---------|---------|--|
| XDCS | XCS | Active low chip select input. A high level forces the serial interface into standby mode, ending the current operation. A high level also forces serial output (SO) to high impedance state. If SM_SDISHARE is 1, pin XDCS is not used, but the signal is generated internally by inverting XCS. |
| | SCK | Serial clock input. The serial clock is also used internally as the master clock for the register interface. SCK can be gated or continuous. In either case, the first rising clock edge after XCS has gone low marks the first bit to be written. |
| | SI | Serial input. If a chip select is active, SI is sampled on the rising CLK edge. |
| - | SO | Serial output. In reads, data is shifted out on the falling SCK edge. In writes SO is at a high impedance state. |

7.1.2 VS1001 Compatibility Mode (deprecated, do not use in new designs)

This mode is active when SM_SDINew is set to 0. In this mode, DCLK, SDATA and BSYNC are active.

| SDI Pin | SCI Pin | Description |
|---------|---------|---|
| - | XCS | Active low chip select input. A high level forces the serial interface into standby mode, ending the current operation. A high level also forces serial output (SO) to high impedance state. |
| BSYNC | - | SDI data is synchronized with a rising edge of BSYNC. |
| DCLK | SCK | Serial clock input. The serial clock is also used internally as the master clock for the register interface. SCK can be gated or continuous. In either case, the first rising clock edge after XCS has gone low marks the first bit to be written. |
| SDATA | SI | Serial input. SI is sampled on the rising SCK edge, if XCS is low. |
| - | SO | Serial output. In reads, data is shifted out on the falling SCK edge. In writes SO is at a high impedance state. |

7.2 Data Request Pin DREQ

The DREQ pin/signal is used to signal if VS1053b's 2048-byte FIFO is capable of receiving data. If DREQ is high, VS1053b can take at least 32 bytes of SDI data or one SCI command. DREQ is turned low when the stream buffer is too full and for the duration of an SCI command.

Because of the 32-byte safety area, the sender may send upto 32 bytes of SDI data at a time without checking the status of DREQ, making controlling VS1053b easier for low-speed microcontrollers.

Note: DREQ may turn low or high at any time, even during a byte transmission. Thus, DREQ should only be used to decide whether to send more bytes. A transmission that has already started doesn't need to be aborted.

Note: In VS1053b DREQ also goes down while an SCI operation is in progress.

There are cases when you still want to send SCI commands when DREQ is low. Because DREQ is shared between SDI and SCI, you can not determine if an SCI command has been executed if SDI is not ready to receive data. In this case you need a long enough delay after every SCI command to make certain none of them are missed. The SCI Registers table in Chapter 9.6 gives the worst-case handling time for each SCI register write.

Note: The status of DREQ can also be read through SCI with the following code. For details on SCI registers, see Chapter 7.4.

```
// This example reads status of DREQ pin through the SPI/SCI register
// interface.
#define SCI_WRAMADDR 7
#define SCI_WRAM 6
while (!endOfFile) {
    int dreq;
    WriteSciReg(SCI_WRAMADDR, 0xC012); // Send address of DREQ register
    dreq = ReadSciReg(SCI_WRAM) & 1; // Read value of DREQ (in bit 0)
    if (dreq) {
        // DREQ high: send 1-32 bytes audio data
    } else {
        // DREQ low: wait 5 milliseconds (so that VS10xx doesn't get
        // continuous SCI operations)
    }
} /* while (!endOfFile) */
```

7.3 Serial Protocol for Serial Data Interface (SPI / SDI)

The serial data interface operates in slave mode so DCLK signal must be generated by an external circuit.

Data (SDATA signal) can be clocked in at either the rising or falling edge of DCLK (Chapter 9.6).

VS1053b assumes its data input to be byte-synchronized. SDI bytes may be transmitted either MSb or LSb first, depending of register SCI_MODE bit SM_SDIORD (Chapter 9.6.1).

The firmware is able to accept the maximum bitrate the SDI supports.

7.3.1 SDI in VS10xx Native Modes (New Mode, recommended)

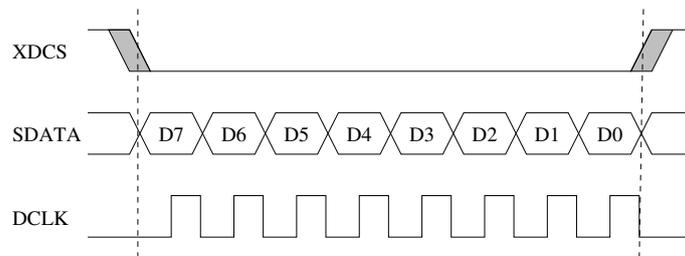


Figure 4: SDI in VS10xx Native Mode, single-byte transfer

In VS10xx native modes (SM_NEWMODE is 1), byte synchronization is achieved by XDCS, as shown in Figure 4. The state of XDCS may not change while a data byte transfer is in progress. XDCS does not need to be deactivated and reactivated for every byte transfer, as shown in Figure 5. However, to maintain data synchronization even if there are occasional clock glitches, it is recommended to deactivate and reactivate XDCS every now and then, for example after each 32 bytes of data.

Note that when sending data through SDI you have to check the Data Request Pin DREQ at least after every 32 bytes (Chapter 7.2).

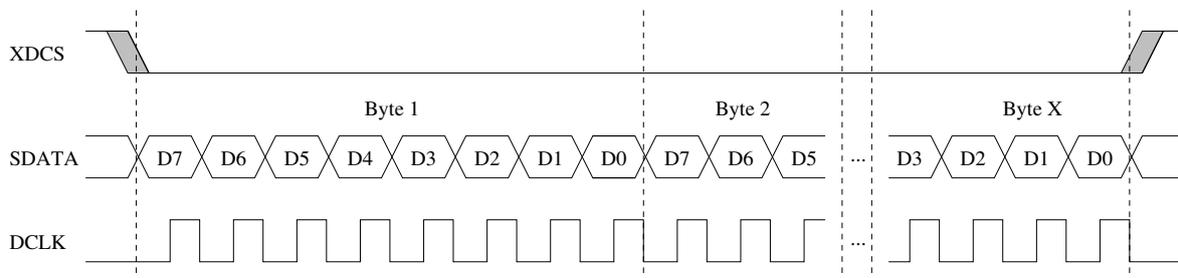


Figure 5: SDI in VS10xx Native Mode, multi-byte transfer, $X \geq 1$

If SM_SDISHARE is 1, the XDCS signal is internally generated by inverting the XCS input.

7.3.2 SDI Timing Diagram in VS10xx Native Modes (New Mode)

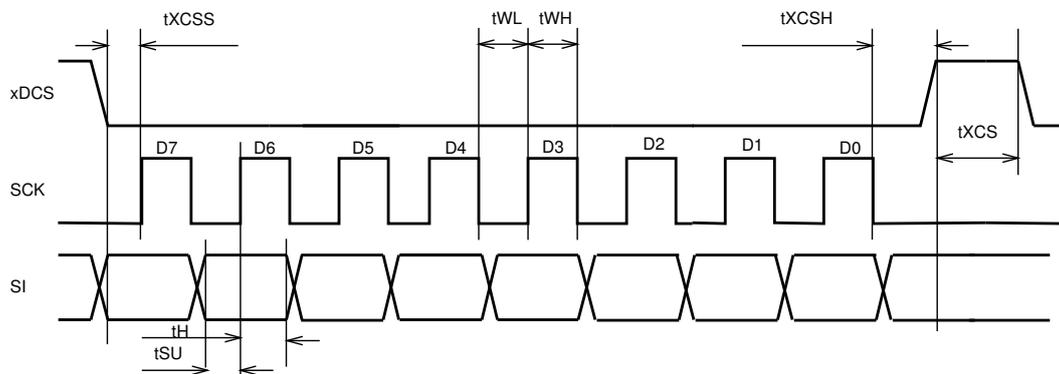


Figure 6: SDI timing diagram

Figure 6 presents SDI bus timing.

| Symbol | Min | Max | Unit |
|--------|-----|-----|-------------|
| tXCSS | 5 | | ns |
| tSU | 0 | | ns |
| tH | 2 | | CLKI cycles |
| tWL | 2 | | CLKI cycles |
| tWH | 2 | | CLKI cycles |
| tXCSH | 1 | | CLKI cycles |
| tXCS | 0 | | CLKI cycles |

Note: xDCS is not required to go high between bytes, so tXCS is 0.

Note: Although the timing is derived from the internal clock CLKI, the system always starts up in 1.0× mode, thus CLKI=XTALI. After you have configured a higher clock through SCI_CLOCKF and waited for DREQ to rise, you can use a higher SPI speed as well.

7.3.3 SDI in VS1001 Compatibility Mode (deprecated, do not use in new designs)

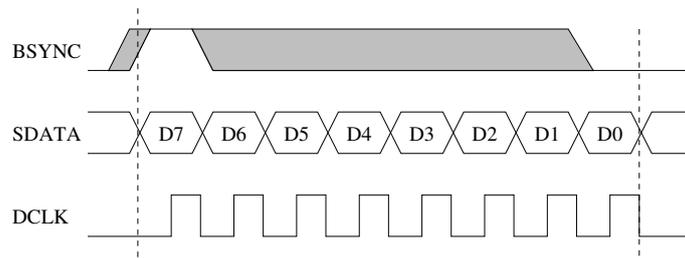


Figure 7: SDI in VS1001 Mode - one byte transfer. Do not use in new designs!

When VS1053b is running in VS1001 compatibility mode, a BSYNC signal must be generated to ensure correct bit-alignment of the input bitstream, as shown in Figures 7 and 8.

The first DCLK sampling edge (rising or falling, depending on selected polarity), during which the BSYNC is high, marks the first bit of a byte (LSB, if LSB-first order is used, MSB, if MSB-first order is used). If BSYNC is '1' when the last bit is received, the receiver stays active and next 8 bits are also received.

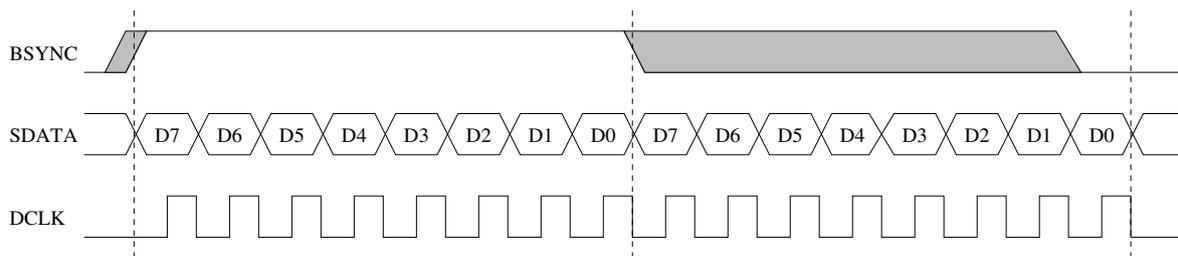


Figure 8: SDI in VS1001 Mode - two byte transfer. Do not use in new designs!

7.3.4 Passive SDI Mode (deprecated, do not use in new designs)

If SM_NEWMODE is 0 and SM_SDISHARE is 1, the operation is otherwise like the VS1001 compatibility mode, but bits are only received while the BSYNC signal is '1'. Rising edge of BSYNC is still used for synchronization.

7.4 Serial Protocol for Serial Command Interface (SPI / SCI)

The serial bus protocol for the Serial Command Interface SCI (Chapter 9.5) consists of an instruction byte, address byte and one 16-bit data word. Each read or write operation can read or write a single register. Data bits are read at the rising edge, so the user should update data at the falling edge. Bytes are always send MSb first. XCS should be low for the full duration of the operation, but you can have pauses between bits if needed.

The operation is specified by an 8-bit instruction opcode. The supported instructions are read and write. See table below.

| Instruction | | |
|-------------|-------------|------------|
| Name | Opcode | Operation |
| READ | 0b0000 0011 | Read data |
| WRITE | 0b0000 0010 | Write data |

Note: VS1053b sets DREQ low after each SCI operation. The duration depends on the operation. It is not allowed to finish a new SCI/SDI operation before DREQ is high again.

7.4.1 SCI Read

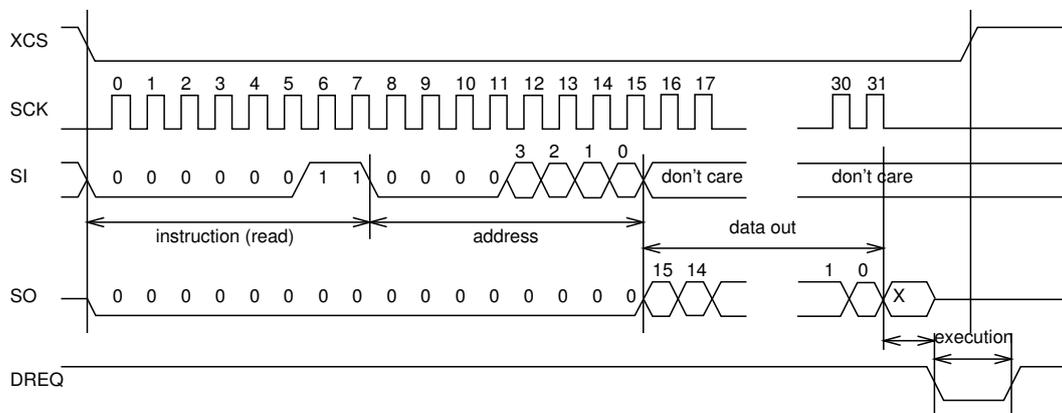


Figure 9: SCI word read

VS1053b registers are read from using the following sequence, as shown in Figure 9. First, XCS line is pulled low to select the device. Then the READ opcode (0x3) is transmitted via the SI line followed by an 8-bit word address. After the address has been read in, any further data on SI is ignored by the chip. The 16-bit data corresponding to the received address will be shifted out onto the SO line.

XCS should be driven high after data has been shifted out.

DREQ is driven low for a short while when in a read operation by the chip. This is a very short time and doesn't require special user attention.

7.4.2 SCI Write

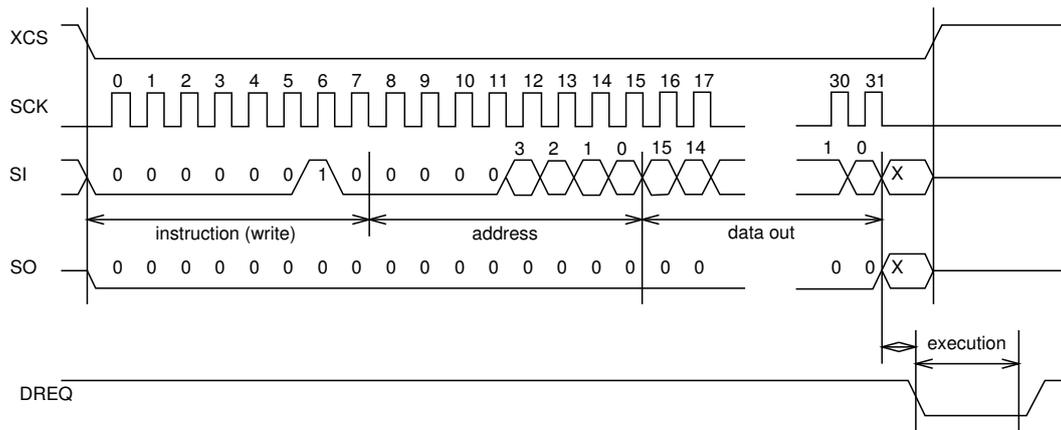


Figure 10: SCI word write

VS1053b registers are written from using the following sequence, as shown in Figure 10. First, XCS line is pulled low to select the device. Then the WRITE opcode (0x2) is transmitted via the SI line followed by an 8-bit word address.

After the word has been shifted in and the last clock has been sent, XCS should be pulled high to end the WRITE sequence.

After the last bit has been sent, DREQ is driven low for the duration of the register update, marked "execution" in the figure. The time varies depending on the register and its contents (see table in Chapter 9.6 for details). If the maximum time is longer than what it takes from the microcontroller to feed the next SCI command or SDI byte, status of DREQ must be checked before finishing the next SCI/SDI operation.

7.4.3 SCI Multiple Write

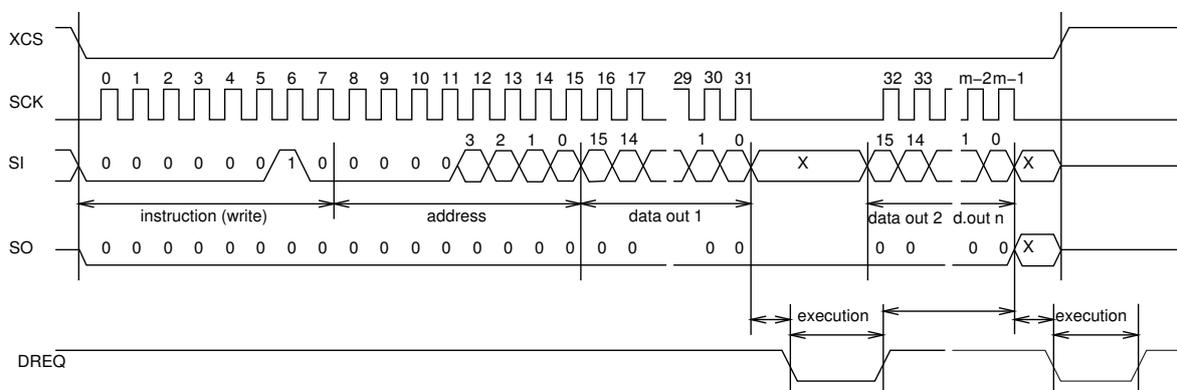


Figure 11: SCI multiple word write

VS1053b allows for the user to send multiple words to the same SCI register, which allows fast SCI uploads, shown in Figure 11. The main difference to a single write is that instead of

bringing XCS up after sending the last bit of a data word, the next data word is sent immediately. After the last data word, XCS is driven high as with a single word write.

After the last bit of a word has been sent, DREQ is driven low for the duration of the register update, marked “execution” in the figure. The time varies depending on the register and its contents (see table in Chapter 9.6 for details). If the maximum time is longer than what it takes from the microcontroller to feed the next SCI command or SDI byte, status of DREQ must be checked before finishing the next SCI/SDI operation.

7.4.4 SCI Timing Diagram

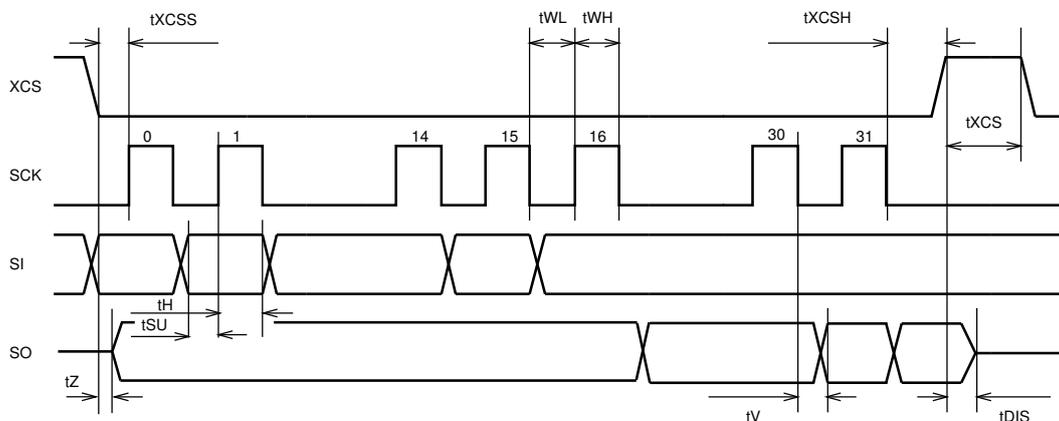


Figure 12: SPI timing diagram

The SCI timing diagram is presented in Figure 12.

| Symbol | Min | Max | Unit |
|--------|---------------------------|-----|-------------|
| tXCSS | 5 | | ns |
| tSU | 0 | | ns |
| tH | 2 | | CLKI cycles |
| tZ | 0 | | ns |
| tWL | 2 | | CLKI cycles |
| tWH | 2 | | CLKI cycles |
| tV | 2 (+ 25 ns ¹) | | CLKI cycles |
| tXCSH | 1 | | CLKI cycles |
| tXCS | 2 | | CLKI cycles |
| tDIS | | 10 | ns |

¹ 25 ns is when pin loaded with 100 pF capacitance. The time is shorter with lower capacitance.

Note: Although the timing is derived from the internal clock CLKI, the system always starts up in 1.0× mode, thus CLKI=XTALI. After you have configured a higher clock through SCI_CLOCKF and waited for DREQ to rise, you can use a higher SPI speed as well.

Note: Because tWL + tWH + tH is 6×CLKI + 25 ns, the maximum speed for SCI reads is CLKI/7.

7.5 SPI Examples with SM_SDINew and SM_SDISHARED set

7.5.1 Two SCI Writes

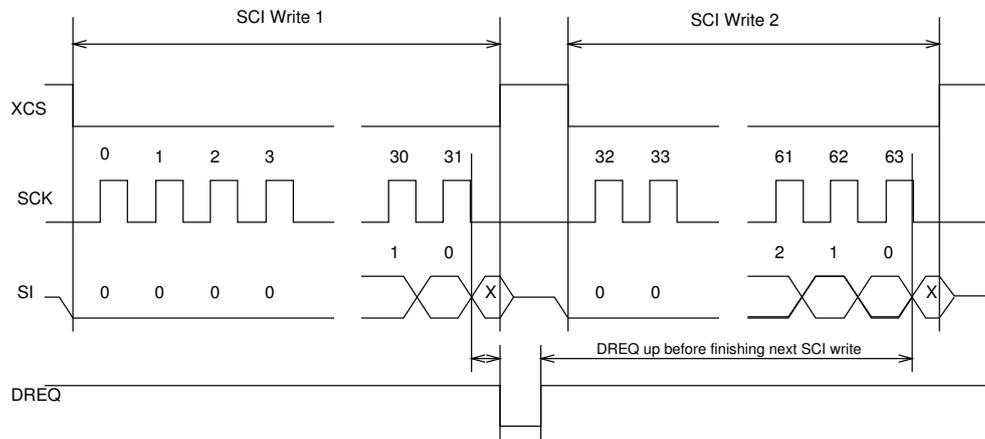


Figure 13: Two SCI operations

Figure 13 shows two consecutive SCI operations. Note that xCS *must* be raised to inactive state between the writes. Also DREQ must be respected as shown in the figure.

7.5.2 Two SDI Bytes

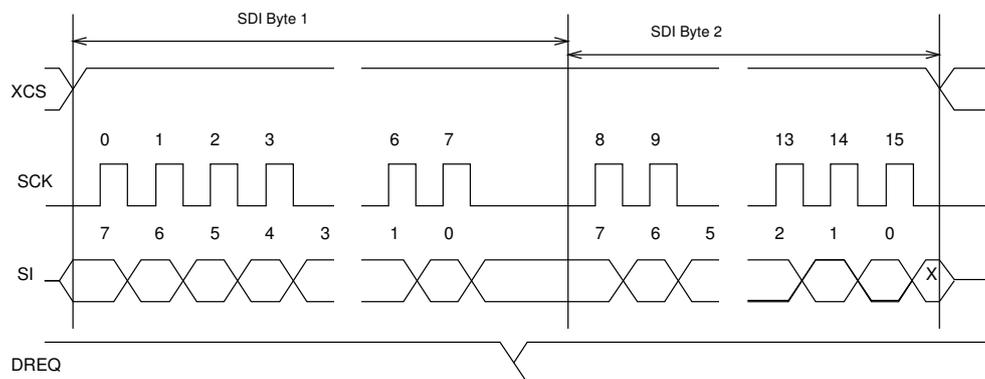


Figure 14: Two SDI bytes

SDI data is synchronized with a raising edge of xCS as shown in Figure 14. However, every byte doesn't need separate synchronization.

7.5.3 SCI Operation in Middle of Two SDI Bytes

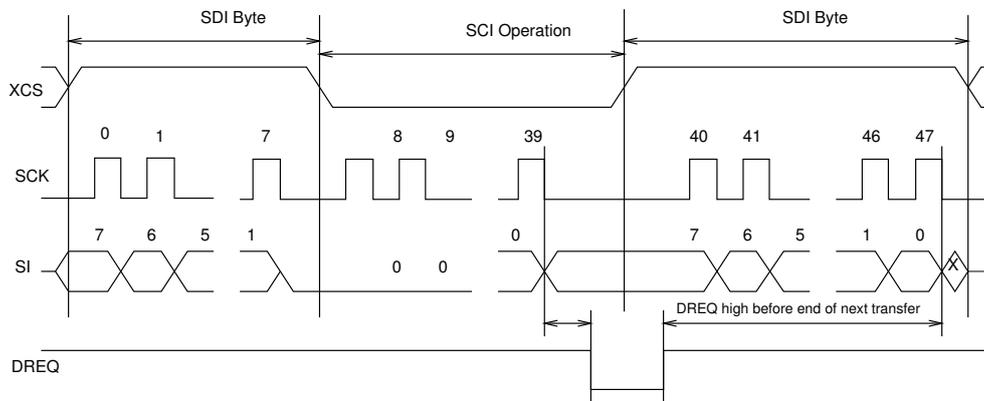


Figure 15: Two SDI bytes separated by an SCI operation

Figure 15 shows how an SCI operation is embedded in between SDI operations. xCS edges are used to synchronize both SDI and SCI. Remember to respect DREQ as shown in the figure.

8 Supported Audio Decoder Formats

| Conventions | |
|-------------|---|
| Mark | Description |
| + | Format is supported |
| ? | Format is supported but not thoroughly tested |
| - | Format exists but is not supported |
| | Format doesn't exist |

8.1 Supported MP3 (MPEG layer III) Formats

MPEG 1.0¹:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | |
|-----------------|------------------|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|
| | 32 | 40 | 48 | 56 | 64 | 80 | 96 | 112 | 128 | 160 | 192 | 224 | 256 | 320 |
| 48000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 44100 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 32000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

MPEG 2.0¹:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | |
|-----------------|------------------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|
| | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 80 | 96 | 112 | 128 | 144 | 160 |
| 24000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 22050 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 16000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

MPEG 2.5¹:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | |
|-----------------|------------------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|
| | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 80 | 96 | 112 | 128 | 144 | 160 |
| 12000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 11025 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 8000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

¹ Also all variable bitrate (VBR) formats are supported.

8.2 Supported MP2 (MPEG layer II) Formats

Note: Layer I / II decoding must be specifically enabled from register SCI_MODE.

MPEG 1.0:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | |
|-----------------|------------------|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|
| | 32 | 48 | 56 | 64 | 80 | 96 | 112 | 128 | 160 | 192 | 224 | 256 | 320 | 384 |
| 48000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 44100 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 32000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

MPEG 2.0:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | |
|-----------------|------------------|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|
| | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 80 | 96 | 112 | 128 | 144 | 160 |
| 24000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 22050 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |
| 16000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

8.3 Supported MP1 (MPEG layer I) Formats

Note: Layer I / II decoding must be specifically enabled from register SCI_MODE.

MPEG 1.0:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | | |
|-----------------|------------------|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|--|
| | 32 | 64 | 96 | 128 | 160 | 192 | 224 | 256 | 288 | 320 | 352 | 384 | 416 | 448 | |
| 48000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + | |
| 44100 | + | + | + | + | + | + | + | + | + | + | + | + | + | + | |
| 32000 | + | + | + | + | + | + | + | + | + | + | + | + | + | + | |

MPEG 2.0:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | | |
|-----------------|------------------|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|--|
| | 32 | 48 | 56 | 64 | 80 | 96 | 112 | 128 | 144 | 160 | 176 | 192 | 224 | 256 | |
| 24000 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | |
| 22050 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | |
| 16000 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | |

8.4 Supported Ogg Vorbis Formats

| Parameter | Min | Max | Unit |
|-------------|-----|-------|----------|
| Channels | | 2 | |
| Window size | 64 | 4096 | samples |
| Samplerate | | 48000 | Hz |
| Bitrate | | 500 | kbit/sec |

Only floor 1 is supported. No known current encoder uses floor 0. All one- and two-channel Ogg Vorbis files should be playable with this decoder.

8.5 Supported AAC (ISO/IEC 13818-7 and ISO/IEC 14496-3) Formats

VS1053b decodes MPEG2-AAC-LC-2.0.0.0 and MPEG4-AAC-LC-2.0.0.0 streams, i.e. the low complexity profile with maximum of two channels can be decoded. If a stream contains more than one element and/or element type, you can select which one to decode from the 16 single-channel, 16 channel-pair, and 16 low-frequency elements. The default is to select the first one that appears in the stream.

Dynamic range control (DRC) is supported and can be controlled by the user to limit or enhance the dynamic range of the material that contains DRC information.

Both Sine window and Kaiser-Bessel-derived window are supported. For MPEG4 pseudo-random noise substitution (PNS) is supported. Short frames (120 and 960 samples) are not supported.

Spectral Band Replication (SBR) level 3, and Parametric Stereo (PS) level 3 are supported (HE-AAC v2). Level 3 means that maximum of 2 channels, samplerates upto and including 48 kHz without and with SBR (with or without PS) are supported. Also, both mixing modes (R_a and R_b), IPD/OPD synthesis and 34 frequency bands resolution are implemented. The downsampled synthesis mode (core coder rates > 24 kHz and ≤ 48 kHz with SBR) is implemented.

SBR and PS decoding can also be disabled. Also different operating modes can be selected. See `config1` and `sbrAndPsStatus` in section 10.11 : "Extra parameters".

If enabled, the internal clock (CLKI) is automatically increased if AAC decoding needs a higher clock. PS and SBR operation is automatically switched off if the internal clock is too slow for correct decoding. Generally HE-AAC v2 files need $4.5\times$ clock to decode both SBR and PS content. This is why $3.5\times + 1.0\times$ clock is the recommended default.

For AAC the streaming ADTS format is recommended. This format allows easy rewind and fast forward because resynchronization is easily possible.

In addition to ADTS (.aac), MPEG2 ADIF (.aac) and MPEG4 AUDIO (.mp4 / .m4a) files are played, but these formats are less suitable for rewind and fast forward operations. You can still implement these features by using the safe jump points table, or using slightly less robust but much easier automatic resync mechanism (see Section 10.5.4).

Because 3GPP (.3gp) and 3GPPv2 (.3g2) files are just MPEG4 files, those that contain only HE-AAC or HE-AACv2 content are played.

Note: To be able to play the .3gp, .3g2, .mp4 and .m4a files, the **mdat** atom must be the last atom in the MP4 file. Because VS1053b receives all data as a stream, all metadata must be available before the music data is received. Several MP4 file formatters do not satisfy this requirement and some kind of conversion is required. This is also why the streamable ADTS format is recommended.

Programs exist that optimize the .mp4 and .m4a into so-called *streamable* format that has the

mdat atom last in the file, and thus suitable for web servers' audio streaming. You can use this kind of tool to process files for VS1053b too. For example `mp4creator -optimize file.mp4`.

AAC^{1,2}:

| Samplerate / Hz | Maximum Bitrate kbit/s - for 2 channels | | | | | | | | |
|-----------------|---|-----|-----|-----|-----|-----|-----|-----|-----|
| | ≤96 | 132 | 144 | 192 | 264 | 288 | 384 | 529 | 576 |
| 48000 | + | + | + | + | + | + | + | + | + |
| 44100 | + | + | + | + | + | + | + | + | |
| 32000 | + | + | + | + | + | + | + | | |
| 24000 | + | + | + | + | + | + | | | |
| 22050 | + | + | + | + | + | | | | |
| 16000 | + | + | + | + | | | | | |
| 12000 | + | + | + | | | | | | |
| 11025 | + | + | | | | | | | |
| 8000 | + | | | | | | | | |

¹ 64000 Hz, 88200 Hz, and 96000 Hz AAC files are played at the highest possible samplerate (48000 Hz with 12.288 MHz XTALI).

² Also all variable bitrate (VBR) formats are supported. Note that the table gives the maximum bitrate allowed for two channels for a specific samplerate as defined by the AAC specification. The decoder does not actually have a fixed lower or upper limit.

8.6 Supported WMA Formats

Windows Media Audio codec versions 2, 7, 8, and 9 are supported. All WMA profiles (L1, L2, and L3) are supported. Previously streams were separated into Classes 1, 2a, 2b, and 3. The decoder has passed Microsoft's conformance testing program. Windows Media Audio Professional is a different codec and is not supported.

WMA 4.0 / 4.1:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | | | | |
|--------------------|------------------|---|---|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|
| | 5 | 6 | 8 | 10 | 12 | 16 | 20 | 22 | 32 | 40 | 48 | 64 | 80 | 96 | 128 | 160 | 192 |
| 8000 | + | + | + | | + | | | | | | | | | | | | |
| 11025 | | | + | + | | | | | | | | | | | | | |
| 16000 | | | | + | + | + | + | | | | | | | | | | |
| 22050 | | | | | | + | + | + | + | | | | | | | | |
| 32000 | | | | | | | + | + | + | + | + | + | | | | | |
| 44100 | | | | | | | | | + | | + | + | + | + | + | + | + |
| 48000 | | | | | | | | | | | | | | | + | + | |

WMA 7:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | | | | |
|--------------------|------------------|---|---|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|
| | 5 | 6 | 8 | 10 | 12 | 16 | 20 | 22 | 32 | 40 | 48 | 64 | 80 | 96 | 128 | 160 | 192 |
| 8000 | + | + | + | | + | | | | | | | | | | | | |
| 11025 | | | + | + | | | | | | | | | | | | | |
| 16000 | | | | + | + | + | + | | | | | | | | | | |
| 22050 | | | | | | + | + | + | + | | | | | | | | |
| 32000 | | | | | | | + | | + | + | + | | | | | | |
| 44100 | | | | | | | | | + | | + | + | + | + | + | + | + |
| 48000 | | | | | | | | | | | | | | | + | + | |

WMA 8:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | | | | |
|--------------------|------------------|---|---|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|
| | 5 | 6 | 8 | 10 | 12 | 16 | 20 | 22 | 32 | 40 | 48 | 64 | 80 | 96 | 128 | 160 | 192 |
| 8000 | + | + | + | | + | | | | | | | | | | | | |
| 11025 | | | + | + | | | | | | | | | | | | | |
| 16000 | | | | + | + | + | + | | | | | | | | | | |
| 22050 | | | | | | + | + | + | + | | | | | | | | |
| 32000 | | | | | | | + | | + | + | + | | | | | | |
| 44100 | | | | | | | | | + | | + | + | + | + | + | + | + |
| 48000 | | | | | | | | | | | | + | + | + | + | + | + |

WMA 9:

| Samplerate / Hz | Bitrate / kbit/s | | | | | | | | | | | | | | | | | | |
|--------------------|------------------|---|---|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|
| | 5 | 6 | 8 | 10 | 12 | 16 | 20 | 22 | 32 | 40 | 48 | 64 | 80 | 96 | 128 | 160 | 192 | 256 | 320 |
| 8000 | + | + | + | | + | | | | | | | | | | | | | | |
| 11025 | | | + | + | | | | | | | | | | | | | | | |
| 16000 | | | | + | + | + | + | | | | | | | | | | | | |
| 22050 | | | | | | + | + | + | + | | | | | | | | | | |
| 32000 | | | | | | | + | | + | + | + | | | | | | | | |
| 44100 | | | | | | | + | | + | | + | + | + | + | + | + | + | + | + |
| 48000 | | | | | | | | | | | | + | + | + | + | + | + | | |

In addition to these expected WMA decoding profiles, all other bitrate and samplerate combinations are supported, including variable bitrate WMA streams. Note that WMA does not consume the bitstream as evenly as MP3, so you need a higher peak transfer capability for clean playback at the same bitrate.

8.7 Supported FLAC Formats

Upto 48 kHz and 24-bit FLAC files are supported with the *VS1053b Patches w/ FLAC Decoder* plugin that is available at <http://www.vlsi.fi/en/support/software/vs10xxplugins.html> . Read the accompanying documentation of the plugin for details.

8.8 Supported RIFF WAV Formats

The most common RIFF WAV subformats are supported, with 1 or 2 audio channels.

| Format | Name | Supported | Comments |
|--------|-------------------|-----------|--|
| 0x01 | PCM | + | 16 and 8 bits, any samplerate \leq 48kHz |
| 0x02 | ADPCM | - | |
| 0x03 | IEEE_FLOAT | - | |
| 0x06 | ALAW | - | |
| 0x07 | MULAW | - | |
| 0x10 | OKI_ADPCM | - | |
| 0x11 | IMA_ADPCM | + | Any samplerate \leq 48kHz |
| 0x15 | DIGISTD | - | |
| 0x16 | DIGIFIX | - | |
| 0x30 | DOLBY_AC2 | - | |
| 0x31 | GSM610 | - | |
| 0x3b | ROCKWELL_ADPCM | - | |
| 0x3c | ROCKWELL_DIGITALK | - | |
| 0x40 | G721_ADPCM | - | |
| 0x41 | G728_CELP | - | |
| 0x50 | MPEG | - | |
| 0x55 | MPEGLAYER3 | + | For supported MP3 modes, see Chapter 8.1 |
| 0x64 | G726_ADPCM | - | |
| 0x65 | G722_ADPCM | - | |

8.9 Supported MIDI Formats

General MIDI and SP-MIDI format 0 files are played. Format 1 and 2 files must be converted to format 0 by the user. The maximum polyphony is 64, the maximum sustained polyphony is 40. Actual polyphony depends on the internal clock rate (which is user-selectable), the instruments used, whether the reverb effect is enabled, and the possible global postprocessing effects enabled, such as bass enhancer, treble control or EarSpeaker spatial processing. The polyphony restriction algorithm makes use of the SP-MIDI MIP table, if present, and uses smooth note removal.

43 MHz ($3.5\times$ input clock) achieves 19-31 simultaneous sustained notes. The instantaneous amount of notes can be larger. This is a fair compromise between power consumption and quality, but higher clocks can be used to increase polyphony.

Reverb effect can be controlled by the user. In addition to reverb automatic and reverb off modes, 14 different decay times can be selected. These roughly correspond to different room sizes. Also, each midi song decides how much effect each instrument gets. Because the reverb effect uses about 4 MHz of processing power the automatic control enables reverb only when the internal clock is at least $3.0\times$.

In VS1053b both EarSpeaker and MIDI reverb can be on simultaneously. This is ideal for listening MIDI songs with headphones.

New instruments have been implemented in addition to the 36 that are available in VS1003. VS1053b now has unique instruments in the whole GM1 instrument set and one bank of GM2 percussions.

Supported MIDI messages:

- meta: 0x51 : set tempo
- other meta: MidiMeta() called
- device control: 0x01 : master volume
- channel message: 0x80 note off, 0x90 note on, 0xc0 program, 0xe0 pitch wheel
- channel message 0xb0: parameter
 - 0x00: bank select (0 is default, 0x78 and 0x7f is drums, 0x79 melodic)
 - 0x06: RPN MSB: 0 = bend range, 2 = coarse tune
 - 0x07: channel volume
 - 0x0a: pan control
 - 0x0b: expression (changes volume)
 - 0x0c: effect control 1 (sets global reverb decay)
 - 0x26: RPN LSB: 0 = bend range
 - 0x40: hold1
 - 0x42: sustenuto
 - 0x5b effects level (channel reverb level)
 - 0x62,0x63,0x64,0x65: NRPN and RPN selects
 - 0x78: all sound off
 - 0x79: reset all controllers
 - 0x7b, 0x7c, 0x7d: all notes off

| VS1053b Melodic Instruments (GM1) | | | |
|-----------------------------------|---------------------------|---------------------------|------------------------|
| 1 Acoustic Grand Piano | 33 Acoustic Bass | 65 Soprano Sax | 97 Rain (FX 1) |
| 2 Bright Acoustic Piano | 34 Electric Bass (finger) | 66 Alto Sax | 98 Sound Track (FX 2) |
| 3 Electric Grand Piano | 35 Electric Bass (pick) | 67 Tenor Sax | 99 Crystal (FX 3) |
| 4 Honky-tonk Piano | 36 Fretless Bass | 68 Baritone Sax | 100 Atmosphere (FX 4) |
| 5 Electric Piano 1 | 37 Slap Bass 1 | 69 Oboe | 101 Brightness (FX 5) |
| 6 Electric Piano 2 | 38 Slap Bass 2 | 70 English Horn | 102 Goblins (FX 6) |
| 7 Harpsichord | 39 Synth Bass 1 | 71 Bassoon | 103 Echoes (FX 7) |
| 8 Clavi | 40 Synth Bass 2 | 72 Clarinet | 104 Sci-fi (FX 8) |
| 9 Celesta | 41 Violin | 73 Piccolo | 105 Sitar |
| 10 Glockenspiel | 42 Viola | 74 Flute | 106 Banjo |
| 11 Music Box | 43 Cello | 75 Recorder | 107 Shamisen |
| 12 Vibraphone | 44 Contrabass | 76 Pan Flute | 108 Koto |
| 13 Marimba | 45 Tremolo Strings | 77 Blown Bottle | 109 Kalimba |
| 14 Xylophone | 46 Pizzicato Strings | 78 Shakuhachi | 110 Bag Pipe |
| 15 Tubular Bells | 47 Orchestral Harp | 79 Whistle | 111 Fiddle |
| 16 Dulcimer | 48 Timpani | 80 Ocarina | 112 Shanai |
| 17 Drawbar Organ | 49 String Ensembles 1 | 81 Square Lead (Lead 1) | 113 Tinkle Bell |
| 18 Percussive Organ | 50 String Ensembles 2 | 82 Saw Lead (Lead) | 114 Agogo |
| 19 Rock Organ | 51 Synth Strings 1 | 83 Calliope Lead (Lead 3) | 115 Pitched Percussion |
| 20 Church Organ | 52 Synth Strings 2 | 84 Chiff Lead (Lead 4) | 116 Woodblock |
| 21 Reed Organ | 53 Choir Aahs | 85 Charang Lead (Lead 5) | 117 Taiko Drum |
| 22 Accordion | 54 Voice Oohs | 86 Voice Lead (Lead 6) | 118 Melodic Tom |
| 23 Harmonica | 55 Synth Voice | 87 Fifths Lead (Lead 7) | 119 Synth Drum |
| 24 Tango Accordion | 56 Orchestra Hit | 88 Bass + Lead (Lead 8) | 120 Reverse Cymbal |
| 25 Acoustic Guitar (nylon) | 57 Trumpet | 89 New Age (Pad 1) | 121 Guitar Fret Noise |
| 26 Acoustic Guitar (steel) | 58 Trombone | 90 Warm Pad (Pad 2) | 122 Breath Noise |
| 27 Electric Guitar (jazz) | 59 Tuba | 91 Polysynth (Pad 3) | 123 Seashore |
| 28 Electric Guitar (clean) | 60 Muted Trumpet | 92 Choir (Pad 4) | 124 Bird Tweet |
| 29 Electric Guitar (muted) | 61 French Horn | 93 Bowed (Pad 5) | 125 Telephone Ring |
| 30 Overdriven Guitar | 62 Brass Section | 94 Metallic (Pad 6) | 126 Helicopter |
| 31 Distortion Guitar | 63 Synth Brass 1 | 95 Halo (Pad 7) | 127 Applause |
| 32 Guitar Harmonics | 64 Synth Brass 2 | 96 Sweep (Pad 8) | 128 Gunshot |

| VS1053b Percussion Instruments (GM1+GM2) | | | |
|--|-------------------------|--------------------------|--------------------------|
| 27 High Q | 43 High Floor Tom | 59 Ride Cymbal 2 | 75 Claves |
| 28 Slap | 44 Pedal Hi-hat [EXC 1] | 60 High Bongo | 76 Hi Wood Block |
| 29 Scratch Push [EXC 7] | 45 Low Tom | 61 Low Bongo | 77 Low Wood Block |
| 30 Scratch Pull [EXC 7] | 46 Open Hi-hat [EXC 1] | 62 Mute Hi Conga | 78 Mute Cuica [EXC 4] |
| 31 Sticks | 47 Low-Mid Tom | 63 Open Hi Conga | 79 Open Cuica [EXC 4] |
| 32 Square Click | 48 High Mid Tom | 64 Low Conga | 80 Mute Triangle [EXC 5] |
| 33 Metronome Click | 49 Crash Cymbal 1 | 65 High Timbale | 81 Open Triangle [EXC 5] |
| 34 Metronome Bell | 50 High Tom | 66 Low Timbale | 82 Shaker |
| 35 Acoustic Bass Drum | 51 Ride Cymbal 1 | 67 High Agogo | 83 Jingle bell |
| 36 Bass Drum 1 | 52 Chinese Cymbal | 68 Low Agogo | 84 Bell tree |
| 37 Side Stick | 53 Ride Bell | 69 Cabasa | 85 Castanets |
| 38 Acoustic Snare | 54 Tambourine | 70 Maracas | 86 Mute Surdo [EXC 6] |
| 39 Hand Clap | 55 Splash Cymbal | 71 Short Whistle [EXC 2] | 87 Open Surdo [EXC 6] |
| 40 Electric Snare | 56 Cowbell | 72 Long Whistle [EXC 2] | |
| 41 Low Floor Tom | 57 Crash Cymbal 2 | 73 Short Guiro [EXC 3] | |
| 42 Closed Hi-hat [EXC 1] | 58 Vibra-slap | 74 Long Guiro [EXC 3] | |

9 Functional Description

9.1 Main Features

VS1053b is based on a proprietary digital signal processor, VS_DSP. It contains all the code and data memory needed for Ogg Vorbis, MP3, AAC, WMA and WAV PCM + ADPCM audio decoding and a MIDI synthesizer, together with serial interfaces, a multirate stereo audio DAC and analog output amplifiers and filters. Also PCM/ADPCM audio encoding is supported using a microphone amplifier and/or line-level inputs and a stereo A/D converter. With software plugins the chip can also decode lossless FLAC as well as record the high-quality Ogg Vorbis format. A UART is provided for debugging purposes.

9.2 Data Flow of VS1053b

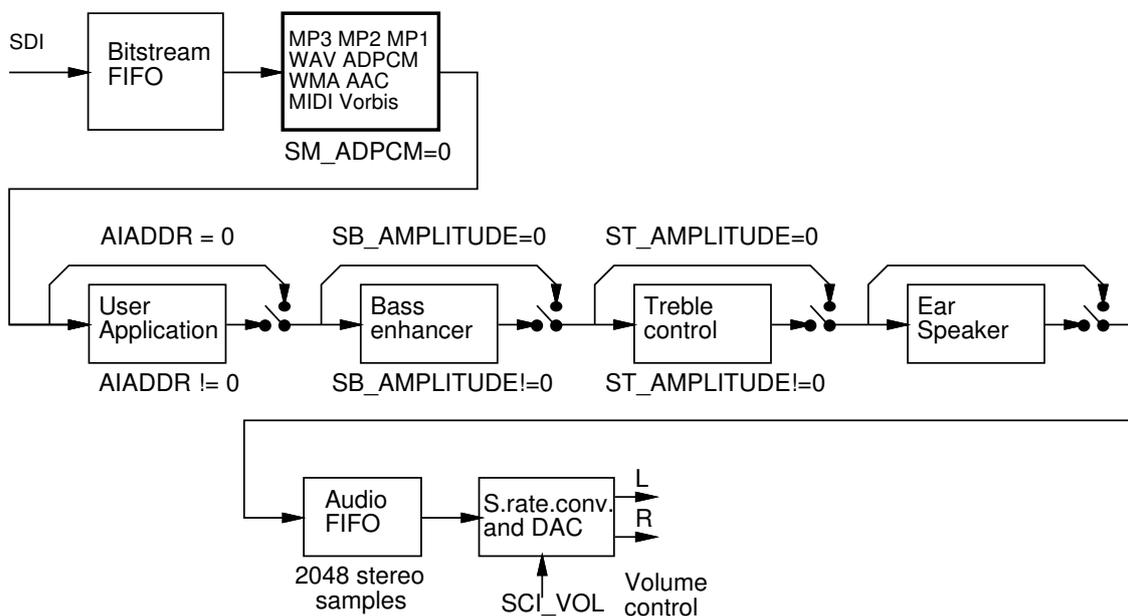


Figure 16: Data flow of VS1053b.

First, depending on the audio data, and provided ADPCM encoding mode is not set, Ogg Vorbis, PCM WAV or IMA ADPCM WAV is received and decoded from the SDI bus.

After decoding, if `SCI_AIADDR` is non-zero, application code is executed from the address pointed to by that register. For more details, see Application Notes for VS10XX.

Then data may be sent to the Bass Enhancer and Treble Control depending on the `SCI_BASS` register.

Next, headphone processing is performed, if the EarSpeaker spatial processing is active.

After that the data to the Audio FIFO, which holds the data until it is read by the Audio interrupt and fed to the samplerate converter and DACs. The size of the audio FIFO is 2048 stereo (2×16 -bit) samples, or 8 KiB.

The samplerate converter upsamples all different samplerates to $XTALI/2$, or 128 times the highest usable samplerate with 18-bit precision. Volume control is performed in the upsampled domain. New volume settings are loaded only when the upsampled signal crosses the zero point (or after a timeout). This zero-crossing detection almost completely removes all audible noise that occurs when volume is suddenly changed.

The samplerate conversion to a common samplerate removes the need for complex PLL-based clocking schemes and allows almost unlimited sample rate accuracy with one fixed input clock frequency. With a 12.288 MHz clock, the DA converter operates at 128×48 kHz, i.e. 6.144 MHz, and creates a stereo in-phase analog signal. The oversampled output is low-pass filtered by an on-chip analog filter. This signal is then forwarded to the earphone amplifier.

9.3 EarSpeaker Spatial Processing

While listening to headphones the sound has a tendency to be localized inside the head. The sound field becomes flat and lacking the sensation of dimensions. This is an unnatural, awkward and sometimes even disturbing situation. This phenomenon is often referred in literature as 'lateralization', meaning 'in-the-head' localization. Long-term listening to lateralized sound may lead to listening fatigue.

All real-life sound sources are external, leaving traces to the acoustic wavefront that arrives to the ear drums. From these traces, the auditory system of the brain is able to judge the distance and angle of each sound source. In loudspeaker listening the sound is external and these traces are available. In headphone listening these traces are missing or ambiguous.

EarSpeaker processes sound to make listening via headphones more like listening to the same music from real loudspeakers or live music. Once EarSpeaker processing is activated, the instruments are moved from inside to the outside of the head, making it easier to separate the different instruments (see figure 17). The listening experience becomes more natural and pleasant, and the stereo image is sharper as the instruments are widely on front of the listener instead of being inside the head.

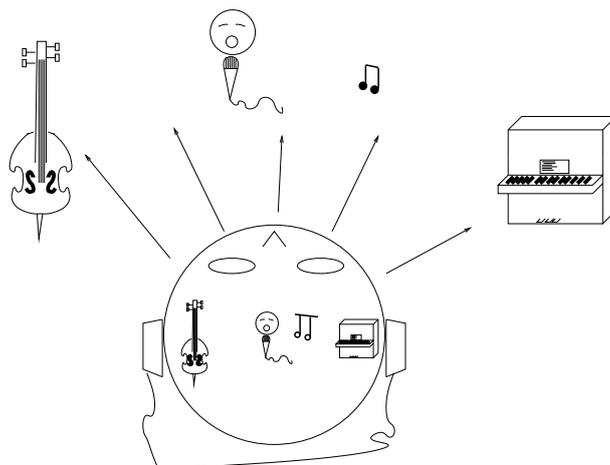


Figure 17: EarSpeaker externalized sound sources vs. normal inside-the-head sound

Note that EarSpeaker differs from any common spatial processing effects, such as echo, reverb, or bass boost. EarSpeaker accurately simulates the human auditory model and real listening environment acoustics. Thus it does not change the tonal character of the music by introducing artificial effects.

EarSpeaker processing can be parameterized to a few different modes, each simulating a little different type of acoustical situation, suiting different personal preferences and types of recording. See section 9.6.1 for how to activate different modes.

- *Off*: Best option when listening through loudspeakers or if the audio to be played contains binaural preprocessing.
- *minimal*: Suited for listening to normal musical scores with headphones, very subtle.

- *normal*: Suited for listening to normal musical scores with headphones, moves sound source further away than *minimal*.
- *extreme*: Suited for old or 'dry' recordings, or if the audio to be played is artificial, for example generated MIDI.

9.4 Serial Data Interface (SDI)

The serial data interface is meant for transferring compressed data for the different decoders of VS1053b.

If the input of the decoder is invalid or it is not received fast enough, analog outputs are automatically muted.

Also several different tests may be activated through SDI as described in Chapter 10.

9.5 Serial Control Interface (SCI)

The serial control interface is compatible with the SPI bus specification. Data transfers are always 16 bits. VS1053b is controlled by writing and reading the registers of the interface.

The main controls of the serial control interface are:

- control of the operation mode, clock, and builtin effects
- access to status information and header data
- receiving encoded data in recording mode
- uploading and controlling user programs

9.6 SCI Registers

VS1053b sets DREQ low when it detects an SCI operation (this delay is 16 to 40 CLKI cycles depending on whether an interrupt service routine is active) and restores it when it has processed the operation. The duration depends on the operation. If DREQ is low when an SCI operation is performed, it also stays low after SCI operation processing.

If DREQ is high before a SCI operation, do not start a new SCI/SDI operation before DREQ is high again. If DREQ is low before a SCI operation because the SDI can not accept more data, make certain there is enough time to complete the operation before sending another.

| SCI registers, prefix SCI_ | | | | | |
|----------------------------|------|---------------------|-------------------------|--------------|---------------------------------|
| Reg | Type | Reset | Time ¹ | Abbrev[bits] | Description |
| 0x0 | rw | 0x4000 ⁶ | 80 CLKI ⁴ | MODE | Mode control |
| 0x1 | rw | 0x000C ³ | 80 CLKI | STATUS | Status of VS1053b |
| 0x2 | rw | 0 | 80 CLKI | BASS | Built-in bass/treble control |
| 0x3 | rw | 0 | 1200 XTALI ⁵ | CLOCKF | Clock freq + multiplier |
| 0x4 | rw | 0 | 100 CLKI | DECODE_TIME | Decode time in seconds |
| 0x5 | rw | 0 | 450 CLKI ² | AUDATA | Misc. audio data |
| 0x6 | rw | 0 | 100 CLKI | WRAM | RAM write/read |
| 0x7 | rw | 0 | 100 CLKI | WRAMADDR | Base address for RAM write/read |
| 0x8 | r | 0 | 80 CLKI | HDATA0 | Stream header data 0 |
| 0x9 | r | 0 | 80 CLKI | HDATA1 | Stream header data 1 |
| 0xA | rw | 0 | 210 CLKI ² | AIADDR | Start address of application |
| 0xB | rw | 0 | 80 CLKI | VOL | Volume control |
| 0xC | rw | 0 | 80 CLKI ² | AICTRL0 | Application control register 0 |
| 0xD | rw | 0 | 80 CLKI ² | AICTRL1 | Application control register 1 |
| 0xE | rw | 0 | 80 CLKI ² | AICTRL2 | Application control register 2 |
| 0xF | rw | 0 | 80 CLKI ² | AICTRL3 | Application control register 3 |

¹ This is the worst-case time that DREQ stays low after writing to this register. The user may choose to skip the DREQ check for those register writes that take less than 100 clock cycles to execute and use a fixed delay instead.

² In addition, the cycles spent in the user application routine must be counted.

³ Firmware changes the value of this register immediately to 0x48 (analog enabled), and after a short while to 0x40 (analog drivers enabled).

⁴ When mode register write specifies a software reset the worst-case time is 22000 XTALI cycles.

⁵ If the clock multiplier is changed, writing to CLOCKF register may force internal clock to run at $1.0 \times XTALI$ for a while. Thus it is not a good idea to send SCI or SDI bits while this register update is in progress.

⁶ Firmware changes the value of this register immediately to 0x4800.

Reads from all SCI registers complete in under 100 CLKI cycles, except a read from AIADDR in 200 cycles. In addition the cycles spent in the user application routine must be counted to the read time of AIADDR, AUDATA, and AICTRL0..3.

9.6.1 SCI_MODE (RW)

SCI_MODE is used to control the operation of VS1053b and defaults to 0x4800 (SM_SDINEW set).

| Bit | Name | Function | Value | Description |
|-----|------------------|------------------------------|-------|-----------------------|
| 0 | SM_DIFF | Differential | 0 | normal in-phase audio |
| | | | 1 | left channel inverted |
| 1 | SM_LAYER12 | Allow MPEG layers I & II | 0 | no |
| | | | 1 | yes |
| 2 | SM_RESET | Soft reset | 0 | no reset |
| | | | 1 | reset |
| 3 | SM_CANCEL | Cancel decoding current file | 0 | no |
| | | | 1 | yes |
| 4 | SM_EARSPEAKER_LO | EarSpeaker low setting | 0 | off |
| | | | 1 | active |
| 5 | SM_TESTS | Allow SDI tests | 0 | not allowed |
| | | | 1 | allowed |
| 6 | SM_STREAM | Stream mode | 0 | no |
| | | | 1 | yes |
| 7 | SM_EARSPEAKER_HI | EarSpeaker high setting | 0 | off |
| | | | 1 | active |
| 8 | SM_DACT | DCLK active edge | 0 | rising |
| | | | 1 | falling |
| 9 | SM_SDIORD | SDI bit order | 0 | MSb first |
| | | | 1 | MSb last |
| 10 | SM_SDISHARE | Share SPI chip select | 0 | no |
| | | | 1 | yes |
| 11 | SM_SDINEW | VS1002 native SPI modes | 0 | no |
| | | | 1 | yes |
| 12 | SM_ADPCM | PCM/ADPCM recording active | 0 | no |
| | | | 1 | yes |
| 13 | - | - | 0 | right |
| | | | 1 | wrong |
| 14 | SM_LINE1 | MIC / LINE1 selector | 0 | MICP |
| | | | 1 | LINE1 |
| 15 | SM_CLK_RANGE | Input clock range | 0 | 12..13 MHz |
| | | | 1 | 24..26 MHz |

When SM_DIFF is set, the player inverts the left channel output. For a stereo input this creates virtual surround, and for a mono input this creates a differential left/right signal.

SM_LAYER12 enables MPEG 1.0 and 2.0 layer I and II decoding in addition to layer III. **If you enable Layer I and Layer II decoding, you are liable for any patent issues that may arise.** Joint licensing of MPEG 1.0 / 2.0 Layer III does not cover all patents pertaining to layers I and II.

Software reset is initiated by setting SM_RESET to 1. This bit is cleared automatically.

If you want to stop decoding a in the middle, set SM_CANCEL, and continue sending data honouring DREQ. When SM_CANCEL is detected by a codec, it will stop decoding and return to the main loop. The stream buffer content is discarded and the SM_CANCEL bit cleared. SCI_HDAT1 will also be cleared. See Chapter 10.5.2 for details.

Bits SM_EARSPEAKER_LO and SM_EARSPEAKER_HI control the EarSpeaker spatial processing. If both are 0, the processing is not active. Other combinations activate the processing and select 3 different effect levels: LO = 1, HI = 0 selects *minimal*, LO = 0, HI = 1 selects *normal*, and LO = 1, HI = 1 selects *extreme*. EarSpeaker takes approximately 12 MIPS at 44.1 kHz samplerate.

If SM_TESTS is set, SDI tests are allowed. For more details on SDI tests, look at Chapter 10.12.

SM_STREAM activates VS1053b's stream mode. In this mode, data should be sent with as even intervals as possible and preferable in blocks of less than 512 bytes, and VS1053b makes every attempt to keep its input buffer half full by changing its playback speed upto 5%. For best quality sound, the average speed error should be within 0.5%, the bitrate should not exceed 160 kbit/s and VBR should not be used. For details, see Application Notes for VS10XX. This mode only works with MP3 and WAV files.

SM_DACT defines the active edge of data clock for SDI. When '0', data is read at the rising edge, when '1', data is read at the falling edge.

When SM_SDIORD is clear, bytes on SDI are sent MSb first. By setting SM_SDIORD, the user may reverse the bit order for SDI, i.e. bit 0 is received first and bit 7 last. Bytes are, however, still sent in the default order. This register bit has no effect on the SCI bus.

Setting SM_SDISHARE makes SCI and SDI share the same chip select, as explained in Chapter 7.1, if also SM_SDINEW is set.

Setting SM_SDINEW will activate VS1002 native serial modes as described in Chapters 7.1.1 and 7.3.1. Note, that this bit is set as a default when VS1053b is started up.

By activating SM_ADPCM and SM_RESET at the same time, the user will activate IMA ADPCM recording mode (see section 10.8).

SM_LINE_IN is used to select the left-channel input for ADPCM recording. If '0', differential microphone input pins MICP and MICN are used; if '1', line-level MICP/LINEIN1 pin is used.

SM_CLK_RANGE activates a clock divider in the XTAL input. When SM_CLK_RANGE is set, the clock is divided by 2 at the input. From the chip's point of view e.g. 24 MHz becomes 12 MHz. SM_CLK_RANGE should be set as soon as possible after a chip reset.

9.6.2 SCI_STATUS (RW)

SCI_STATUS contains information on the current status of VS1053b. It also controls some low-level things that the user does not usually have to care about.

| Name | Bits | Description |
|------------------|-------|---|
| SS_DO_NOT_JUMP | 15 | Header in decode, do not fast forward/rewind |
| SS_SWING | 14:12 | Set swing to +0 dB, +0.5 dB, ..., or +3.5 dB |
| SS_VCM_OVERLOAD | 11 | GBUF overload indicator '1' = overload |
| SS_VCM_DISABLE | 10 | GBUF overload detection '1' = disable |
| | 9:8 | reserved |
| SS_VER | 7:4 | Version |
| SS_APDOWN2 | 3 | Analog driver powerdown |
| SS_APDOWN1 | 2 | Analog internal powerdown |
| SS_AD_CLOCK | 1 | AD clock select, '0' = 6 MHz, '1' = 3 MHz |
| SS_REFERENCE_SEL | 0 | Reference voltage selection, '0' = 1.23 V, '1' = 1.65 V |

SS_DO_NOT_JUMP is set when a WAV, Ogg Vorbis, WMA, MP4, or AAC-ADIF header is being decoded and jumping to another location in the file is not allowed. If you use soft reset or cancel, clear this bit yourself or it can be accidentally left set.

If AVDD is at least 3.3 V, SS_REFERENCE_SEL can be set to select 1.65 V reference voltage to increase the analog output swing.

SS_AD_CLOCK can be set to divide the AD modulator frequency by 2 if XTALI/2 is too much.

SS_VER is 0 for VS1001, 1 for VS1011, 2 for VS1002, 3 for VS1003, 4 for VS1053 and VS8053, 5 for VS1033, 7 for VS1103, and 6 for VS1063.

SS_APDOWN2 controls analog driver powerdown. SS_APDOWN1 controls internal analog powerdown. These bit are meant to be used by the system firmware only.

If the user wants to powerdown VS1053b with a minimum power-off transient, set SCI_VOL to 0xffff, then wait for at least a few milliseconds before activating reset.

VS1053b contains GBUF protection circuit which disconnects the GBUF driver when too much current is drawn, indicating a short-circuit to ground. SS_VCM_OVERLOAD is high while the overload is detected. SS_VCM_DISABLE can be set to disable the protection feature.

SS_SWING allows you to go above the 0 dB volume setting. Value 0 is normal mode, 1 gives +0.5 dB, and 2 gives +1.0 dB. Settings from 3 to 7 cause the DAC modulator to be overdriven and should not be used. You can use SS_SWING with I2S to control the amount of headroom.

Note: Due to a firmware bug in the VS1053b volume calculation routine clears SS_AD_CLOCK and SS_REFERENCE_SEL bits. Write to SCI_STATUS or SCI_VOLUME, and sample rate change (if bass enhancer or treble control are active) causes the volume calculation routine to be called. See the *VS1053b Patches w/ FLAC Decoder* plugin for a workaround:

<http://www.vlsi.fi/en/support/software/vs10xxplugins.html>

9.6.3 SCI_BASS (RW)

| Name | Bits | Description |
|--------------|-------|---|
| ST_AMPLITUDE | 15:12 | Treble Control in 1.5 dB steps (-8..7, 0 = off) |
| ST_FREQLIMIT | 11:8 | Lower limit frequency in 1000 Hz steps (1..15) |
| SB_AMPLITUDE | 7:4 | Bass Enhancement in 1 dB steps (0..15, 0 = off) |
| SB_FREQLIMIT | 3:0 | Lower limit frequency in 10 Hz steps (2..15) |

The Bass Enhancer VSBE is a powerful bass boosting DSP algorithm, which tries to take the most out of the users earphones without causing clipping.

VSBE is activated when SB_AMPLITUDE is non-zero. SB_AMPLITUDE should be set to the user's preferences, and SB_FREQLIMIT to roughly 1.5 times the lowest frequency the user's audio system can reproduce. For example setting SCI_BASS to 0x00f6 will have 15 dB enhancement below 60 Hz.

Note: Because VSBE tries to avoid clipping, it gives the best bass boost with dynamical music material, or when the playback volume is not set to maximum. It also does not create bass: the source material must have some bass to begin with.

Treble Control VSTC is activated when ST_AMPLITUDE is non-zero. For example setting SCI_BASS to 0x7a00 will have 10.5 dB treble enhancement at and above 10 kHz.

Bass Enhancer uses about 2.1 MIPS and Treble Control 1.2 MIPS at 44100 Hz samplerate. Both can be on simultaneously.

In VS1053b bass and treble initialization and volume change is delayed until the next batch of samples are sent to the audio FIFO. Thus, unlike with earlier VS10XX chips, audio interrupts can no longer be missed when SCI_BASS or SCI_VOL is written to.

9.6.4 SCI_CLOCKF (RW)

The external clock multiplier SCI register SCI_CLOCKF, which has changed slightly since VS1003 and VS1033, is presented in the table below.

| SCI_CLOCKF bits | | |
|-----------------|-------|-----------------------------|
| Name | Bits | Description |
| SC_MULT | 15:13 | Clock multiplier |
| SC_ADD | 12:11 | Allowed multiplier addition |
| SC_FREQ | 10: 0 | Clock frequency |

SC_MULT activates the built-in clock multiplier. This will multiply XTALI to create a higher CLKI. When the multiplier is changed by more than 0.5×, the chip runs at 1.0× clock for a few hundred clock cycles. The values are as follows:

| SC_MULT | MASK | CLKI |
|---------|--------|-----------|
| 0 | 0x0000 | XTALI |
| 1 | 0x2000 | XTALI×2.0 |
| 2 | 0x4000 | XTALI×2.5 |
| 3 | 0x6000 | XTALI×3.0 |
| 4 | 0x8000 | XTALI×3.5 |
| 5 | 0xa000 | XTALI×4.0 |
| 6 | 0xc000 | XTALI×4.5 |
| 7 | 0xe000 | XTALI×5.0 |

SC_ADD tells how much the decoder firmware is allowed to add to the multiplier specified by SC_MULT if more cycles are temporarily needed to decode a WMA or AAC stream. The values are:

| SC_ADD | MASK | Multiplier addition |
|--------|--------|----------------------------|
| 0 | 0x0000 | No modification is allowed |
| 1 | 0x0800 | XTALI×1.0 |
| 2 | 0x1000 | XTALI×1.5 |
| 3 | 0x1800 | XTALI×2.0 |

If SC_FREQ is non-zero, it tells that the input clock XTALI is running at something else than 12.288 MHz. XTALI is set in 4 kHz steps. The formula for calculating the correct value for this register is $\frac{XTALI - 8000000}{4000}$ (XTALI is in Hz).

Note: because maximum samplerate is $\frac{XTALI}{256}$, all samplerates are not available if XTALI < 12.288 MHz.

Note: Automatic clock change can only happen when decoding WMA and AAC files. Automatic clock change is done one 0.5× at a time. This does not cause a drop to 1.0× clock and you can use the same SCI and SDI clock throughout the file.

Example: If SCI_CLOCKF is 0x8BE8, SC_MULT = 4, SC_ADD = 1 and SC_FREQ = 0x3E8 = 1000. This means that XTALI = 1000 × 4000 + 8000000 = 12 MHz. The clock multiplier is set to 3.5×XTALI = 42 MHz, and the maximum allowed multiplier that the firmware may automatically choose to use is (3.5 + 1.0)×XTALI = 54 MHz.

9.6.5 SCI_DECODE_TIME (RW)

When decoding correct data, current decoded time is shown in this register in full seconds.

The user may change the value of this register. In that case the new value should be written twice to make absolutely certain that the change is not overwritten by the firmware. A write to SCI_DECODE_TIME also resets the `byteRate` calculation.

SCI_DECODE_TIME is reset at every hardware and software reset. It is no longer cleared when decoding of a file ends to allow the decode time to proceed automatically with looped files and with seamless playback of multiple files.

With fast playback (see the `playSpeed` extra parameter) the decode time also counts faster.

Some codecs (WMA and Ogg Vorbis) can also indicate the absolute play position, see the `positionMsec` extra parameter in section 10.11.

9.6.6 SCI_AUDATA (RW)

When decoding correct data, the current samplerate and number of channels can be found in bits 15:1 and 0 of SCI_AUDATA, respectively. Bits 15:1 contain the samplerate divided by two, and bit 0 is 0 for mono data and 1 for stereo. Writing to SCI_AUDATA will change the samplerate directly.

Example: 44100 Hz stereo data reads as 0xAC45 (44101).

Example: 11025 Hz mono data reads as 0x2B10 (11024).

Example: Writing 0xAC80 sets samplerate to 44160 Hz, stereo mode does not change.

To reduce digital power consumption when idle, you can write a low samplerate to SCI_AUDATA.

Note: Ogg Vorbis decoding overrides AUDATA change. If you want to fine-tune samplerate in streaming applications with Ogg Vorbis, use SCI_CLOCKF to control the playback rate instead of AUDATA.

9.6.7 SCI_WRAM (RW)

SCI_WRAM is used to upload application programs and data to instruction and data RAMs. The start address must be initialized by writing to SCI_WRAMADDR prior to the first write/read of SCI_WRAM. As 16 bits of data can be transferred with one SCI_WRAM write/read, and the instruction word is 32 bits long, two consecutive writes/reads are needed for each instruction word. The byte order is big-endian (i.e. most significant words first). After each full-word write/read, the internal pointer is autoincremented.

9.6.8 SCI_WRAMADDR (W)

SCI_WRAMADDR is used to set the program address for following SCI_WRAM writes/reads. Use an address offset from the following table to access X, Y, I or peripheral memory.

| WRAMADDR Start... End | Dest. addr. Start... End | Bits/ Word | Description |
|--------------------------|-----------------------------|---------------|-----------------|
| 0x1800... 0x18XX | 0x1800... 0x18XX | 16 | X data RAM |
| 0x5800... 0x58XX | 0x1800... 0x18XX | 16 | Y data RAM |
| 0x8040... 0x84FF | 0x0040... 0x04FF | 32 | Instruction RAM |
| 0xC000... 0xFFFF | 0xC000... 0xFFFF | 16 | I/O |

Only user areas in X, Y, and instruction memory are listed above. Other areas can be accessed, but should not be written to unless otherwise specified.

9.6.9 SCI_HDAT0 and SCI_HDAT1 (R)

For WAV files, SCI_HDAT1 contains 0x7665 (“ve”). SCI_HDAT0 contains the data rate measured in bytes per second for all supported RIFF WAVE formats: mono and stereo 8-bit or 16-bit PCM, mono and stereo IMA ADPCM. To get the bitrate of the file, multiply the value by 8.

For AAC ADTS streams, SCI_HDAT1 contains 0x4154 (“AT”). For AAC ADIF files, SCI_HDAT1 contains 0x4144 (“AD”). For AAC .mp4 / .m4a files, SCI_HDAT1 contains 0x4D34 (“M4”). SCI_HDAT0 contains the average data rate in bytes per second. To get the bitrate of the file, multiply the value by 8.

For WMA files, SCI_HDAT1 contains 0x574D (“WM”) and SCI_HDAT0 contains the data rate measured in bytes per second. To get the bitrate of the file, multiply the value by 8.

For MIDI files, SCI_HDAT1 contains 0x4D54 (“MT”) and SCI_HDAT0 contains the average data rate in bytes per second. To get the bitrate of the file, multiply the value by 8.

For Ogg Vorbis files, SCI_HDAT1 contains 0x4F67 “Og”. SCI_HDAT0 contains the average data rate in bytes per second. To get the bitrate of the file, multiply the value by 8.

For MP3 files, SCI_HDAT1 is between 0xFFE0 and 0xFFFF. SCI_HDAT1 / 0 contain the following:

| Bit | Function | Value | Explanation |
|--------------|-------------|-------|--------------------------------|
| HDAT1[15:5] | syncword | 2047 | stream valid |
| HDAT1[4:3] | ID | 3 | ISO 11172-3 MPG 1.0 |
| | | 2 | ISO 13818-3 MPG 2.0 (1/2-rate) |
| | | 1 | MPG 2.5 (1/4-rate) |
| | | 0 | MPG 2.5 (1/4-rate) |
| HDAT1[2:1] | layer | 3 | I |
| | | 2 | II |
| | | 1 | III |
| | | 0 | reserved |
| HDAT1[0] | protect bit | 1 | No CRC |
| | | 0 | CRC protected |
| HDAT0[15:12] | bitrate | | see bitrate table |
| HDAT0[11:10] | samplerate | 3 | reserved |
| | | 2 | 32/16/ 8 kHz |
| | | 1 | 48/24/12 kHz |
| | | 0 | 44/22/11 kHz |
| HDAT0[9] | pad bit | 1 | additional slot |
| | | 0 | normal frame |
| HDAT0[8] | private bit | | not defined |
| HDAT0[7:6] | mode | 3 | mono |
| | | 2 | dual channel |
| | | 1 | joint stereo |
| | | 0 | stereo |
| HDAT0[5:4] | extension | | see ISO 11172-3 |
| HDAT0[3] | copyright | 1 | copyrighted |
| | | 0 | free |
| HDAT0[2] | original | 1 | original |
| | | 0 | copy |
| HDAT0[1:0] | emphasis | 3 | CCITT J.17 |
| | | 2 | reserved |
| | | 1 | 50/15 microsec |
| | | 0 | none |

When read, SCI_HDAT0 and SCI_HDAT1 contain header information that is extracted from MP3 stream currently being decoded. After reset both registers are cleared, indicating no data has been found yet.

The “samplerate” field in SCI_HDAT0 is interpreted according to the following table:

| “samplerate” | ID=3 | ID=2 | ID=0,1 |
|--------------|-------|-------|--------|
| 3 | - | - | - |
| 2 | 32000 | 16000 | 8000 |
| 1 | 48000 | 24000 | 12000 |
| 0 | 44100 | 22050 | 11025 |

The “bitrate” field in HDAT0 is read according to the following table. Notice that for variable bitrate stream the value changes constantly.

| "bitrate" | Layer I | | Layer II | | Layer III | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | ID=3 | ID=0,1,2 | ID=3 | ID=0,1,2 | ID=3 | ID=0,1,2 |
| | kbit/s | | kbit/s | | kbit/s | |
| 15 | forbidden | forbidden | forbidden | forbidden | forbidden | forbidden |
| 14 | 448 | 256 | 384 | 160 | 320 | 160 |
| 13 | 416 | 224 | 320 | 144 | 256 | 144 |
| 12 | 384 | 192 | 256 | 128 | 224 | 128 |
| 11 | 352 | 176 | 224 | 112 | 192 | 112 |
| 10 | 320 | 160 | 192 | 96 | 160 | 96 |
| 9 | 288 | 144 | 160 | 80 | 128 | 80 |
| 8 | 256 | 128 | 128 | 64 | 112 | 64 |
| 7 | 224 | 112 | 112 | 56 | 96 | 56 |
| 6 | 192 | 96 | 96 | 48 | 80 | 48 |
| 5 | 160 | 80 | 80 | 40 | 64 | 40 |
| 4 | 128 | 64 | 64 | 32 | 56 | 32 |
| 3 | 96 | 56 | 56 | 24 | 48 | 24 |
| 2 | 64 | 48 | 48 | 16 | 40 | 16 |
| 1 | 32 | 32 | 32 | 8 | 32 | 8 |
| 0 | - | - | - | - | - | - |

The average data rate in bytes per second can be read from memory, see the `byteRate` extra parameter. This variable contains the byte rate for all codecs. To get the bitrate of the file, multiply the value by 8.

The bitrate calculation is not automatically reset between songs, but it can also be reset without a software or hardware reset by writing to `SCI_DECODE_TIME`.

9.6.10 SCI_AIADDR (RW)

`SCI_AIADDR` indicates the start address of the application code written earlier with `SCI_WRAMADDR` and `SCI_WRAM` registers. If no application code is used, this register should not be initialized, or it should be initialized to zero. For more details, see Application Notes for VS10XX.

Note: Reading `AIADDR` is not recommended. It can cause `samplerate` to be set to a very low value.

9.6.11 SCI_VOL (RW)

SCI_VOL is a volume control for the player hardware. The most significant byte of the volume register controls the left channel volume, the low part controls the right channel volume. The channel volume sets the attenuation from the maximum volume level in 0.5 dB steps. Thus, maximum volume is 0x0000 and total silence is 0xFEFE.

Note, that after hardware reset the volume is set to full volume. Resetting the software does not reset the volume setting.

Setting SCI_VOL to 0xFFFF will activate analog powerdown mode.

Example: for a volume of -2.0 dB for the left channel and -3.5 dB for the right channel: $(2.0/0.5) = 4$, $3.5/0.5 = 7 \rightarrow \text{SCI_VOL} = 0x0407$.

Example: $\text{SCI_VOL} = 0x2424 \rightarrow$ both left and right volumes are $0x24 * -0.5 = -18.0 \text{ dB}$

In VS1053b bass and treble initialization and volume change is delayed until the next batch of samples are sent to the audio FIFO. Thus, audio interrupts can no longer be missed during a write to SCI_BASS or SCI_VOL.

This delays the volume setting slightly, but because the volume control is now done in the DAC hardware instead of performing it to the samples going into the audio FIFO, the overall volume change response is better than before. Also, the actual volume control has zero-cross detection, which almost completely removes all audible noise that occurs when volume is suddenly changed.

9.6.12 SCI_AICTRL[x] (RW)

SCI_AICTRL[x] registers ($x=[0 \dots 3]$) can be used to access the user's application program.

The AICTRL registers are also used with PCM/ADPCM encoding mode.

10 Operation

10.1 Clocking

VS1053b operates on a single, nominally 12.288 MHz fundamental frequency master clock. This clock can be generated by external circuitry (connected to pin XTALI) or by the internal clock crystal interface (pins XTALI and XTALO). This clock is used by the analog parts and determines the highest available samplerate. With 12.288 MHz clock all samplerates upto 48000 Hz are available.

VS1053b can also use 24..26 MHz clocks when SM_CLK_RANGE in the SCI_MODE register is set to 1. The system clock is then divided by 2 at the clock input and the chip gets a 12..13 MHz input clock.

10.2 Hardware Reset

When the XRESET -signal is driven low, VS1053b is reset and all the control registers and internal states are set to the initial values. XRESET-signal is asynchronous to any external clock. The reset mode doubles as a full-powerdown mode, where both digital and analog parts of VS1053b are in minimum power consumption stage, and where clocks are stopped. Also XTALO is grounded.

When XRESET is asserted, all output pins go to their default states. All input pins will go to high-impedance state (to input state), except SO, which is still controlled by the XCS.

After a hardware reset (or at power-up) DREQ will stay down for around 22000 clock cycles, which means an approximate 1.8 ms delay if VS1053b is run at 12.288 MHz. After this the user should set such basic software registers as SCI_MODE, SCI_BASS, SCI_CLOCKF, and SCI_VOL before starting decoding. See section 9.6 for details.

If the input clock is 24..26 MHz, SM_CLK_RANGE should be set as soon as possible after a chip reset without waiting for DREQ.

Internal clock can be multiplied with a PLL. Supported multipliers through the SCI_CLOCKF register are $1.0 \times \dots 5.0 \times$ the input clock. Reset value for Internal Clock Multiplier is $1.0 \times$. If typical values are wanted, the Internal Clock Multiplier needs to be set to $3.5 \times$ after reset. Wait until DREQ rises, then write value 0x9800 to SCI_CLOCKF (register 3). See section 9.6.4 for details.

10.3 Software Reset

In some cases the decoder software has to be reset. This is done by activating bit SM_RESET in register SCI_MODE (Chapter 9.6.1). Then wait for at least 2 μ s, then look at DREQ. DREQ

will stay down for about 22000 clock cycles, which means an approximate 1.8 ms delay if VS1053b is run at 12.288 MHz. After DREQ is up, you may continue playback as usual.

As opposed to all earlier VS10XX chips, it is not recommended to do a software reset between songs. This way the user may be sure that even files with low samplersates or bitrates are played right to their end.

10.4 Low Power Mode

If you need to keep the system running while not decoding data, but need to lower the power consumption, you can use the following tricks.

- Select the 1.0× clock by writing 0x0000 to SCI_CLOCKF. This disables the PLL and saves some power.
- Write a low non-zero value, such as 0x0010 to SCI_AUDATA. This will reduce the samplerate and the number of audio interrupts required. Between audio interrupts the VSDSP core will just wait for an interrupt, thus saving power.
- Turn off all audio post-processing (tone controls and EarSpeaker).
- If possible for the application, write 0xffff to SCI_VOL to disable the analog drivers.

To return from low-power mode, revert register values in reverse order.

Note: The low power mode consumes significantly more electricity than hardware reset.

10.5 Play and Decode

This is the normal operation mode of VS1053b. SDI data is decoded. Decoded samples are converted to analog domain by the internal DAC. If no decodable data is found, SCI_HDAT0 and SCI_HDAT1 are set to 0.

When there is no input for decoding, VS1053b goes into idle mode (lower power consumption than during decoding) and actively monitors the serial data input for valid data.

10.5.1 Playing a Whole File

This is the default playback mode.

1. Send an audio file to VS1053b.
2. Read extra parameter value `endFillByte` (Chapter 10.11).
3. Send at least 2052 bytes of `endFillByte[7:0]`.
4. Set `SCI_MODE` bit `SM_CANCEL`.
5. Send at least 32 bytes of `endFillByte[7:0]`.
6. Read `SCI_MODE`. If `SM_CANCEL` is still set, go to 5. If `SM_CANCEL` hasn't cleared after sending 2048 bytes, do a software reset (this should be extremely rare).
7. The song has now been successfully sent. `HDAT0` and `HDAT1` should now both contain 0 to indicate that no format is being decoded. Return to 1.

10.5.2 Cancelling Playback

Cancelling playback of a song is a normal operation when the user wants to jump to another song while doing playback.

1. Send a portion of an audio file to VS1053b.
2. Set `SCI_MODE` bit `SM_CANCEL`.
3. Continue sending audio file, but check `SM_CANCEL` after every 32 bytes of data. If it is still set, goto 3. If `SM_CANCEL` doesn't clear after 2048 bytes or one second, do a software reset (this should be extremely rare).
4. When `SM_CANCEL` has cleared, read extra parameter value `endFillByte` (Chapter 10.11).
5. Send 2052 bytes of `endFillByte[7:0]`.
6. `HDAT0` and `HDAT1` should now both contain 0 to indicate that no format is being decoded. You can now send the next audio file.

10.5.3 Fast Play

VS1053b allows fast audio playback. If your microcontroller can feed data fast enough to the VS1053b, this is the preferred way to fast forward audio.

1. Start sending an audio file to VS1053b.
2. To set fast play, set extra parameter value `playSpeed` (Chapter 10.11).
3. Continue sending audio file.
4. To exit fast play mode, write 1 to `playSpeed`.

To estimate whether or not your microcontroller can feed enough data to VS1053b in fast play mode, see contents of extra parameter value `byteRate` (Chapter 10.11). Note that `byteRate` contains the data speed of the file played back at nominal speed even when fast play is active.

Note: Play speed is not reset when song is changed.

10.5.4 Fast Forward and Rewind without Audio

To do fast forward and rewind you need the capability to do random access to the audio file. Unfortunately fast forward and rewind isn't available at all times, like when file headers are being read.

1. Send a portion of an audio file to VS1053b.
2. When random access is required, read SCI_STATUS bit SS_DO_NOT_JUMP. If that bit is set, random access cannot be performed, so go back to 1.
3. Read extra parameter value endFillByte (Chapter 10.11).
4. Send at least 2048 bytes of endFillByte[7:0].
5. Jump forwards or backwards in the file.
6. Continue sending the file.

Note: It is recommended that playback volume is decreased by e.g. 10 dB when fast forwarding/rewinding.

Note: Register DECODE_TIME does not take jumps into account.

Note: Midi is not suitable for random-access. You can implement fast forward using the playSpeed extra parameter to select 1-128× play speed. SCI_DECODE_TIME also speeds up. If necessary, rewind can be implemented by restarting decoding of a MIDI file and fast playing to the appropriate place. SCI_DECODE_TIME can be used to decide when the right place has been reached.

10.5.5 Maintaining Correct Decode Time

When fast forward and rewind operations are performed, there is no way to maintain correct decode time for most files. However, WMA and Ogg Vorbis files offer exact time information in the file. To use accurate time information whenever possible, use the following algorithm:

1. Start sending an audio file to VS1053b.
2. Read extra parameter value pair positionMsec (Chapter 10.11).
3. If positionMsec is -1, show you estimation of decoding time using DECODE_TIME (and your estimate of file position if you have performed fast forward / rewind operations).
4. If positionMsec is not -1, use this time to show the exact position in the file.

10.6 Feeding PCM data

VS1053b can be used as a PCM decoder by sending a WAV file header. If the length sent in the WAV header is 0xFFFFFFFF, VS1053b will stay in PCM mode indefinitely (or until SM_CANCEL has been set). 8-bit linear and 16-bit linear audio is supported in mono or stereo. A WAV header looks like this:

| File Offset | Field Name | Size | Bytes | Description |
|-------------|---------------|------|---------------------|---|
| 0 | ChunkID | 4 | "RIFF" | |
| 4 | ChunkSize | 4 | 0xff 0xff 0xff 0xff | |
| 8 | Format | 4 | "WAVE" | |
| 12 | SubChunk1ID | 4 | "fmt " | |
| 16 | SubChunk1Size | 4 | 0x10 0x0 0x0 0x0 | 16 |
| 20 | AudioFormat | 2 | 0x1 0x0 | Linear PCM |
| 22 | NumOfChannels | 2 | C0 C1 | 1 for mono, 2 for stereo |
| 24 | SampleRate | 4 | S0 S1 S2 S3 | 0x1f40 for 8 kHz |
| 28 | ByteRate | 4 | R0 R1 R2 R3 | 0x3e80 for 8 kHz 16-bit mono |
| 32 | BlockAlign | 2 | A0 A1 | 0x02 0x00 for mono, 0x04 0x00 for stereo 16-bit |
| 34 | BitsPerSample | 2 | B0 B1 | 0x10 0x00 for 16-bit data |
| 52 | SubChunk2ID | 4 | "data" | |
| 56 | SubChunk2Size | 4 | 0xff 0xff 0xff 0xff | Data size |

The rules to calculate the four variables are as follows:

- S = sample rate in Hz, e.g. 44100 for 44.1 kHz.
- For 8-bit data $B = 8$, and for 16-bit data $B = 16$.
- For mono data $C = 1$, for stereo data $C = 2$.
- $A = \frac{C \times B}{8}$.
- $R = S \times A$.

Example: A 44100 Hz 16-bit stereo PCM header would read as follows:

```
0000 52 49 46 46 ff ff ff ff 57 41 56 45 66 6d 74 20 |RIFF...WAVEfmt |
0100 10 00 00 00 01 00 02 00 44 ac 00 00 10 b1 02 00 |.....D.....|
0200 04 00 10 00 64 61 74 61 ff ff ff ff |....data....|
```

10.7 Ogg Vorbis Recording

Ogg Vorbis is an open file format that allows for very high sound quality with low to medium bitrates.

Ogg Vorbis recording is activated by loading the Ogg Vorbis Encoder Application to the 16 KiB program RAM memory of the VS1053b. After activation, encoder results can be read from registers SCI_HDAT0 and SCI_HDAT1, much like when using PCM/ADPCM recording (Chapter 10.8).

Three profiles are provided: one for high-quality stereo recording at a bitrate of approx. 140 kbit/s, and two for speech-quality mono recording at a bitrates between 15 and 30 kbit/s.

To use the Ogg Vorbis Encoder application, please load the application from VLSI Solution's Web page <http://www.vlsi.fi/en/support/software/vs10xxapplications.html> and read the accompanying documentation.

10.8 PCM/ADPCM Recording

This chapter explains how to create RIFF/WAV file in PCM or IMA ADPCM format. IMA ADPCM is a widely supported ADPCM format and many PC audio playback programs can play it. IMA ADPCM recording gives roughly a compression ratio of 4:1 compared to linear, 16-bit audio. This makes it possible to record for example one 8 kHz audio at 32.44 kbit/s.

VS1053 has a stereo ADC, thus also two-channel (separate AGC, if AGC enabled) and stereo (common AGC, if AGC enabled) modes are available. Mono recording mode selects either left or right channel. Left channel is either MIC or LINE1 depending on the SCI_MODE register.

10.8.1 Activating ADPCM Mode

| Register | Bits | Description |
|-------------|------------------------|---|
| SCI_MODE | 2, 12, 14 | Start ADPCM mode, select MIC/LINE1 |
| SCI_AICTRL0 | 15..0 | Sample rate 8000..48000 Hz (read at recording startup) |
| SCI_AICTRL1 | 15..0 | Recording gain (1024 = 1×) or 0 for automatic gain control |
| SCI_AICTRL2 | 15..0 | Maximum autogain amplification (1024 = 1×, 65535 = 64×) |
| SCI_AICTRL3 | 1..0 2 15..3 | 0 = joint stereo (common AGC), 1 = dual channel (separate AGC), 2 = left channel, 3 = right channel 0 = IMA ADPCM mode, 1 = LINEAR PCM mode reserved, set to 0 |

PCM / IMA ADPCM recording mode is activated by setting bits SM_RESET and SM_ADPCM in SCI_MODE. Line input 1 is used instead of differential mic input if SM_LINE1 is set. Before activating ADPCM recording, user **must** write the right values to SCI_AICTRL0 and SCI_AICTRL3. These values are only read at recording startup. SCI_AICTRL1 and SCI_AICTRL2 can be altered anytime, but it is preferable to write good init values before activation.

SCI_AICTRL1 controls linear recording gain. 1024 is equal to digital gain 1, 512 is equal to digital gain 0.5 and so on. If the user wants to use automatic gain control (AGC), SCI_AICTRL1 should be set to 0. Typical speech applications usually are better off using AGC, as this takes care of relatively uniform speech loudness in recordings.

SCI_AICTRL2 controls the maximum AGC gain. This can be used to limit the amplification of noise when there is no signal. If SCI_AICTRL2 is zero, the maximum gain is initialized to 65535 (64×), i.e. whole range is used.

For example:

```
WriteVS10xxRegister(SCI_AICTRL0, 16000U);
WriteVS10xxRegister(SCI_AICTRL1, 0);
WriteVS10xxRegister(SCI_AICTRL2, 4096U);
WriteVS10xxRegister(SCI_AICTRL3, 0);
WriteVS10xxRegister(SCI_MODE, ReadVS10xxRegister(SCI_MODE) |
                        SM_RESET | SM_ADPCM | SM_LINE1);
WriteVS10xxPatch(); /* Only for VS1053b and VS8053b */
```

selects 16 kHz, stereo mode with automatic gain control and maximum amplification of 4×.

WriteVS10xxPatch() should perform the following SCI writes (only for VS1053b and VS8053b):

| Register | Reg. No | Value |
|--------------|---------|--------|
| SCI_WRAMADDR | 0x7 | 0x8010 |
| SCI_WRAM | 0x6 | 0x3e12 |
| SCI_WRAM | 0x6 | 0xb817 |
| SCI_WRAM | 0x6 | 0x3e14 |
| SCI_WRAM | 0x6 | 0xf812 |
| SCI_WRAM | 0x6 | 0x3e01 |
| SCI_WRAM | 0x6 | 0xb811 |
| SCI_WRAM | 0x6 | 0x0007 |
| SCI_WRAM | 0x6 | 0x9717 |
| SCI_WRAM | 0x6 | 0x0020 |
| SCI_WRAM | 0x6 | 0xffd2 |
| SCI_WRAM | 0x6 | 0x0030 |
| SCI_WRAM | 0x6 | 0x11d1 |
| SCI_WRAM | 0x6 | 0x3111 |
| SCI_WRAM | 0x6 | 0x8024 |
| SCI_WRAM | 0x6 | 0x3704 |
| SCI_WRAM | 0x6 | 0xc024 |
| SCI_WRAM | 0x6 | 0x3b81 |
| SCI_WRAM | 0x6 | 0x8024 |
| SCI_WRAM | 0x6 | 0x3101 |
| SCI_WRAM | 0x6 | 0x8024 |
| SCI_WRAM | 0x6 | 0x3b81 |
| SCI_WRAM | 0x6 | 0x8024 |
| SCI_WRAM | 0x6 | 0x3f04 |
| SCI_WRAM | 0x6 | 0xc024 |
| SCI_WRAM | 0x6 | 0x2808 |
| SCI_WRAM | 0x6 | 0x4800 |
| SCI_WRAM | 0x6 | 0x36f1 |
| SCI_WRAM | 0x6 | 0x9811 |
| SCI_WRAMADDR | 0x7 | 0x8028 |
| SCI_WRAM | 0x6 | 0x2a00 |
| SCI_WRAM | 0x6 | 0x040e |

This patch is also available from VLSI Solution's web page <http://www.vlsi.fi/en/support/software/vs10xxpatches.html> by the name of *VS1053b IMA ADPCM Encoder Fix*, and it is also part of the VS1053b patches package.

10.8.2 Reading PCM / IMA ADPCM Data

After PCM / IMA ADPCM recording has been activated, registers SCI_HDAT0 and SCI_HDAT1 have new functions.

The PCM / IMA ADPCM sample buffer is 1024 16-bit words. The fill status of the buffer can be read from SCI_HDAT1. If SCI_HDAT1 is greater than 0, you can read as many 16-bit words from SCI_HDAT0. If the data is not read fast enough, the buffer overflows and returns to empty state.

Note: if $SCI_HDAT1 \geq 768$, it may be better to wait for the buffer to overflow and clear before reading samples. That way you may avoid buffer aliasing.

In IMA ADPCM mode each mono IMA ADPCM block is 128 words, i.e. 256 bytes, and stereo

IMA ADPCM block is 256 words, i.e. 512 bytes. If you wish to interrupt reading data and possibly continue later, please stop at the boundary. This way whole blocks are skipped and the encoded stream stays valid.

10.8.3 Adding a PCM RIFF Header

To make your PCM file a RIFF / WAV file, you have to add a header to the data. The following shows a header for a mono file. Note that 2- and 4-byte values are little-endian (lowest byte first).

| File Offset | Field Name | Size | Bytes | Description |
|-------------|---------------|------|------------------|--------------------------|
| 0 | ChunkID | 4 | "RIFF" | |
| 4 | ChunkSize | 4 | F0 F1 F2 F3 | File size - 8 |
| 8 | Format | 4 | "WAVE" | |
| 12 | SubChunk1ID | 4 | "fmt " | |
| 16 | SubChunk1Size | 4 | 0x10 0x0 0x0 0x0 | 20 |
| 20 | AudioFormat | 2 | 0x01 0x0 | 0x1 for PCM |
| 22 | NumOfChannels | 2 | C0 C1 | 1 for mono, 2 for stereo |
| 24 | SampleRate | 4 | R0 R1 R2 R3 | 0x1f40 for 8 kHz |
| 28 | ByteRate | 4 | B0 B1 B2 B3 | 0x3e80 for 8 kHz mono |
| 32 | BlockAlign | 2 | 0x02 0x00 | 2 for mono, 4 for stereo |
| 34 | BitsPerSample | 2 | 0x10 0x00 | 16 bits / sample |
| 36 | SubChunk3ID | 4 | "data" | |
| 40 | SubChunk3Size | 4 | D0 D1 D2 D3 | Data size (File Size-36) |
| 44 | Samples... | | | Audio samples |

The values in the table are calculated as follows:

$$R = F_s \text{ (see Chapter 10.8.1 to see how to calculate } F_s \text{)}$$

$$B = 2 \times F_s \times C$$

If you know beforehand how much you are going to record, you may fill in the complete header before any actual data. However, if you don't know how much you are going to record, you have to fill in the header size datas F and D after finishing recording.

The PCM data is read from SCI_HDAT0 and written into file as follows. The high 8 bits of SCI_HDAT0 should be written as the first byte to a file, then the low 8 bits. Note that this is contrary to the default operation of some 16-bit microcontrollers, and you may have to take extra care to do this right.

Below is an example of a valid header for a 44.1 kHz mono PCM file that has a final length of 1798768 (0x1B7270) bytes:

```

0000  52 49 46 46 68 72 1b 00  57 41 56 45 66 6d 74 20  |RIFFhr..WAVEfmt |
0010  10 00 00 00 01 00 01 00  80 bb 00 00 00 77 01 00  |.....w...|
0020  02 00 10 00 64 61 74 61  44 72 1b 00                |....dataDr.....|

```

10.8.4 Adding an IMA ADPCM RIFF Header

To make your IMA ADPCM file a RIFF / WAV file, you have to add a header to the data. The following shows a header for a mono file. Note that 2- and 4-byte values are little-endian (lowest byte first).

| File Offset | Field Name | Size | Bytes | Description |
|-------------|---------------|------|------------------|---|
| 0 | ChunkID | 4 | "RIFF" | |
| 4 | ChunkSize | 4 | F0 F1 F2 F3 | File size - 8 |
| 8 | Format | 4 | "WAVE" | |
| 12 | SubChunk1ID | 4 | "fmt " | |
| 16 | SubChunk1Size | 4 | 0x14 0x0 0x0 0x0 | 20 |
| 20 | AudioFormat | 2 | 0x11 0x0 | 0x11 for IMA ADPCM |
| 22 | NumOfChannels | 2 | C0 C1 | 1 for mono, 2 for stereo |
| 24 | SampleRate | 4 | R0 R1 R2 R3 | 0x1f40 for 8 kHz |
| 28 | ByteRate | 4 | B0 B1 B2 B3 | 0xfd7 for 8 kHz mono |
| 32 | BlockAlign | 2 | 0x00 0x01 | 256 for mono, 512 for stereo |
| 34 | BitsPerSample | 2 | 0x04 0x00 | 4-bit ADPCM |
| 36 | ByteExtraData | 2 | 0x02 0x00 | 2 |
| 38 | ExtraData | 2 | 0xf9 0x01 | Samples per block (505) |
| 40 | SubChunk2ID | 4 | "fact" | |
| 44 | SubChunk2Size | 4 | 0x4 0x0 0x0 0x0 | 4 |
| 48 | NumOfSamples | 4 | S0 S1 S2 S3 | |
| 52 | SubChunk3ID | 4 | "data" | |
| 56 | SubChunk3Size | 4 | D0 D1 D2 D3 | Data size (File Size-60) |
| 60 | Block1 | 256 | | First ADPCM block, 512 bytes for stereo |
| 316 | ... | | | More ADPCM data blocks |

If we have n audio blocks, the values in the table are as follows:

$$F = n \times C \times 256 + 52$$

$$R = F_s \text{ (see Chapter 10.8.1 to see how to calculate } F_s \text{)}$$

$$B = \frac{F_s \times C \times 256}{505}$$

$$S = n \times 505. \quad D = n \times C \times 256$$

If you know beforehand how much you are going to record, you may fill in the complete header before any actual data. However, if you don't know how much you are going to record, you have to fill in the header size datas F , S and D after finishing recording.

The 128 words (256 words for stereo) of an ADPCM block are read from SCI_HDAT0 and written into file as follows. The high 8 bits of SCI_HDAT0 should be written as the first byte to a file, then the low 8 bits. Note that this is contrary to the default operation of some 16-bit microcontrollers, and you may have to take extra care to do this right.

To see if you have written the mono file in the right way check bytes 2 and 3 (the first byte counts as byte 0) of each 256-byte block. Byte 2 should be 0..88 and byte 3 should be zero. For stereo you check bytes 2, 3, 6, and 7 of each 512-byte block. Bytes 2 and 6 should be 0..88. Bytes 3 and 7 should be zero.

Below is an example of a valid header for a 44.1 kHz stereo IMA ADPCM file that has a final length of 10038844 (0x992E3C) bytes:

```

0000 52 49 46 46 34 2e 99 00 57 41 56 45 66 6d 74 20 |RIFF4...WAVEfmt |
0010 14 00 00 00 11 00 02 00 44 ac 00 00 a7 ae 00 00 |.....D.....|
0020 00 02 04 00 02 00 f9 01 66 61 63 74 04 00 00 00 |.....fact....|
0030 14 15 97 00 64 61 74 61 00 2e 99 00                |....data....|

```

10.8.5 Playing ADPCM Data

In order to play back your PCM / IMA ADPCM recordings, you have to have a file with a header as described in Chapter 10.8.3 or Chapter 10.8.4. If this is the case, all you need to do is to provide the ADPCM file through SDI as you would with any audio file.

10.8.6 Sample Rate Considerations

VS10xx chips that support IMA ADPCM playback are capable of playing back ADPCM files with any sample rate. However, some other programs may expect IMA ADPCM files to have some exact sample rates, like 8000 or 11025 Hz. Also, some programs or systems do not support sample rates below 8000 Hz.

If you want better quality with the expense of increased data rate, you can use higher sample rates, for example 16 kHz.

10.8.7 Record Monitoring Volume

In VS1053b writing to the SCI_VOL register during IMA ADPCM encoding does not change the volume. You need to set a suitable volume before activating the IMA ADPCM mode, or you can use the VS1053 hardware volume control register DAC_VOL directly.

For example:

```

WriteVS10xxRegister(SCI_WRAMADDR, 0xc045); /*DAC_VOL*/
WriteVS10xxRegister(SCI_WRAM,      0x0101); /*-6.0 dB*/

```

The hardware volume control DAC_VOL (address 0xc045) allows 0.5 dB steps for both left (high 8 bits) and right channel (low 8 bits). The low 4 bits of both 8-bit values set the attenuation in 6 dB steps, the high 4 bits in 0.5 dB steps.

| dB | DAC_VOL | dB | DAC_VOL | dB | DAC_VOL | dB | DAC_VOL |
|------|---------|-------|---------|-------|---------|-------|---------|
| -0.0 | 0x0000 | -6.5 | 0xb2b2 | : | : | -60.0 | 0x0a0a |
| -0.5 | 0xb1b1 | : | : | -36.0 | 0x0606 | -60.5 | 0xbbbb |
| -1.0 | 0xa1a1 | -12.0 | 0x0202 | -36.5 | 0xb7b7 | : | : |
| -1.5 | 0x9191 | -12.5 | 0xb3b3 | : | : | -66.0 | 0x0b0b |
| -2.0 | 0x8181 | : | : | -42.0 | 0x0707 | -66.5 | 0xbcbc |
| -2.5 | 0x7171 | -18.0 | 0x0303 | -42.5 | 0xb8b8 | : | : |
| -3.0 | 0x6161 | -18.5 | 0xb4b4 | : | : | -72.0 | 0x0c0c |
| -3.5 | 0x5151 | : | : | -48.0 | 0x0808 | -72.5 | 0xbdbd |
| -4.0 | 0x4141 | -24.0 | 0x0404 | -48.5 | 0xb9b9 | : | : |
| -4.5 | 0x3131 | -24.5 | 0xb5b5 | : | : | -78.0 | 0x0d0d |
| -5.0 | 0x2121 | : | : | -54.0 | 0x0909 | -78.5 | 0xbebe |
| -5.5 | 0x1111 | -30.0 | 0x0505 | -54.5 | 0xbaba | : | : |
| -6.0 | 0x0101 | -30.5 | 0xb6b6 | : | : | -84.0 | 0x0e0e |

10.9 SPI Boot

If GPIO0 is set with a pull-up resistor to 1 at boot time, VS1053b tries to boot from external SPI memory.

SPI boot redefines the following pins:

| Normal Mode | SPI Boot Mode |
|-------------|---------------|
| GPIO0 | xCS |
| GPIO1 | CLK |
| DREQ | MOSI |
| GPIO2 | MISO |

The memory has to be an SPI Bus Serial EEPROM with 16-bit or 24-bit addresses. The serial speed used by VS1053b is 245 kHz with the nominal 12.288 MHz clock. The first three bytes in the memory have to be 0x50, 0x26, 0x48.

10.10 Real-Time MIDI

If GPIO0 is low and GPIO1 is high during boot, real-time MIDI mode is activated. In this mode the PLL is configured to 4.0×, the UART is configured to the MIDI data rate 31250 bps, and real-time MIDI data is then read from UART and SDI. Both input methods should not be used simultaneously. If you use SDI, first send 0x00 and then send the MIDI data byte.

EarSpeaker setting can be configured with GPIO2 and GPIO3. The state of GPIO2 and GPIO3 are only read at startup.

Real-Time MIDI can also be started with a small patch code using SCI.

Note: The real-time MIDI parser in VS1053b does not know how to skip SysEx messages. An improved version can be loaded into IRAM if needed.

10.11 Extra Parameters

The following structure is in X memory at address 0x1e02 (note the different location than in VS1033) and can be used to change some extra parameters or get useful information.

```
#define PARAMETRIC_VERSION 0x0003
struct parametric {
    /* configs are not cleared between files */
    u_int16 version; /*1e02 - structure version */
    u_int16 config1; /*1e03 ---- ---- ppss RRRR PS mode, SBR mode, Reverb */
    u_int16 playSpeed; /*1e04 0,1 = normal speed, 2 = twice, 3 = three times etc. */
    u_int16 byteRate; /*1e05 average byterate */

    u_int16 endFillByte; /*1e06 byte value to send after file sent */
    u_int16 reserved[16]; /*1e07..15 file byte offsets */
    u_int32 jumpPoints[8]; /*1e16..25 file byte offsets */
    u_int16 latestJump; /*1e26 index to lastly updated jumpPoint */
    u_int32 positionMsec /*1e27-28 play position, if known (WMA, Ogg Vorbis) */
    s_int16 resync; /*1e29 > 0 for automatic m4a, ADIF, WMA resyncs */
    union {
        struct {
            u_int32 curPacketSize;
            u_int32 packetSize;
        } wma;
        struct {
            u_int16 sceFoundMask; /*1e2a SCE's found since last clear */
            u_int16 cpeFoundMask; /*1e2b CPE's found since last clear */
            u_int16 lfeFoundMask; /*1e2c LFE's found since last clear */
            u_int16 playSelect; /*1e2d 0 = first any, initialized at aac init */
            s_int16 dynCompress; /*1e2e -8192=1.0, initialized at aac init */
            s_int16 dynBoost; /*1e2f 8192=1.0, initialized at aac init */
            u_int16 sbrAndPsStatus; /*0x1e30 1=SBR, 2=upsample, 4=PS, 8=PS active */
        } aac;
        struct {
            u_int32 bytesLeft;
        } midi;
        struct {
            s_int16 gain; /* 0x1e2a proposed gain offset in 0.5dB steps, default = -12 */
        } vorbis;
    } i;
};
```

Notice that reading two-word variables through the SCI_WRAMADDR and SCI_WRAM interface is not protected in any way. The variable can be updated between the read of the low and high parts. The problem arises when both the low and high parts change values. To determine if the value is correct, you should read the value twice and compare the results.

The following example shows what happens when bytesLeft is decreased from 0x10000 to 0xffff and the update happens between low and high part reads or after high part read.

| Read Invalid | | Read Valid | | No Update | |
|--------------|--------------------------|------------|--------------------------|-----------|--------|
| Address | Value | Address | Value | Address | Value |
| 0x1e2a | 0x0000 change after this | 0x1e2a | 0x0000 | 0x1e2a | 0x0000 |
| 0x1e2b | 0x0000 | 0x1e2b | 0x0001 change after this | 0x1e2b | 0x0001 |
| 0x1e2a | 0xffff | 0x1e2a | 0xffff | 0x1e2a | 0x0000 |
| 0x1e2b | 0x0000 | 0x1e2b | 0x0000 | 0x1e2b | 0x0001 |

You can see that in the invalid read the low part wraps from 0x0000 to 0xffff while the high part stays the same. In this case the second read gives a valid answer, otherwise always use the value of the first read. The second read is needed when it is possible that the low part wraps around, changing the high part, i.e. when the low part is small. `bytesLeft` is only decreased by one at a time, so a reread is needed only if the low part is 0.

10.11.1 Common Parameters

These parameters are common for all codecs. Other fields are only valid when the corresponding codec is active. The currently active codec can be determined from `SCI_HDAT1`.

| Parameter | Address | Usage |
|---------------|-----------|---|
| version | 0x1e02 | Structure version – 0x0003 |
| config1 | 0x1e03 | Miscellaneous configuration |
| playSpeed | 0x1e04 | 0,1 = normal speed, 2 = twice, 3 = three times etc. |
| byteRate | 0x1e05 | average byterate |
| endFillByte | 0x1e06 | byte to send after file |
| jumpPoints[8] | 0x1e16-25 | Packet offsets for WMA and AAC |
| latestJump | 0x1e26 | Index to latest jumpPoint |
| positionMsec | 0x1e27-28 | File position in milliseconds, if available |
| resync | 0x1e29 | Automatic resync selector |

The fuse-programmed ID is read at startup and copied into the `chipID` field. If not available, the value will be all zeros. The `version` field can be used to determine the layout of the rest of the structure. The version number is changed when the structure is changed. For VS1053b the structure version is 3.

`config1` controls MIDI Reverb and AAC's SBR and PS settings.

`playSpeed` makes it possible to fast forward songs. Decoding of the bitstream is performed, but only each `playSpeed` frames are played. For example by writing 4 to `playSpeed` will play the song four times as fast as normal, if you are able to feed the data with that speed. Write 0 or 1 to return to normal speed. `SCI_DECODE_TIME` will also count faster. All current codecs support the `playSpeed` configuration.

`byteRate` contains the average bitrate in bytes per second for every code. The value is updated once per second and it can be used to calculate an estimate of the remaining playtime. This value is also available in `SCI_HDAT0` for all codecs except MP3, MP2, and MP1.

`endFillByte` indicates what byte value to send after file is sent before `SM_CANCEL`.

`jumpPoints` contain 32-bit file offsets. Each valid (non-zero) entry indicates a start of a packet for WMA or start of a raw data block for AAC (ADIF, .mp4 / .m4a). `latestJump` contains the index of the entry that was updated last. If you only read entry pointed to by `latestJump` you do *not* need to read the entry twice to ensure validity. Jump point information can be used to

implement perfect fast forward and rewind for WMA and AAC (ADIF, .mp4 / .m4a).

`positionMsec` is a field that gives the current play position in a file in milliseconds, regardless of rewind and fast forward operations. The value is only available in codecs that can determine the play position from the stream itself. Currently WMA and Ogg Vorbis provide this information. If the position is unknown, this field contains -1.

`resync` field is used to force a resynchronization to the stream for WMA and AAC (ADIF, .mp4 / .m4a) instead of ending the decode at first error. This field can be used to implement almost perfect fast forward and rewind for WMA and AAC (ADIF, .mp4 / .m4a). The user should set this field before performing data seeks if they are not in packet or data block boundaries. The field value tells how many tries are allowed before giving up. The value 32767 gives infinite tries.

The `resync` field is set to 32767 after a reset to make resynchronization the default action, but it can be cleared after reset to restore the old action. When `resync` is set, every file decode should always end as described in Chapter 10.5.1.

Seek fields no longer exist. When `resync` is required, WMA and AAC codecs now enter broadcast/stream mode where file size information is ignored. Also, the file size and sample size information of WAV files are ignored when `resync` is non-zero. The user must use `SM_CANCEL` or software reset to end decoding.

Note: WAV, WMA, ADIF, and .mp4 / .m4a files begin with a metadata or header section, which must be fully processed before any fast forward or rewind operation. `SS_DO_NOT_JUMP` (in `SCI_STATUS`) is clear when the header information has been processed and jumps are allowed.

10.11.2 WMA

| Parameter | Address | Usage |
|----------------------------|-----------|--|
| <code>curPacketSize</code> | 0x1e2a/2b | The size of the packet being processed |
| <code>packetSize</code> | 0x1e2c/2d | The packet size in ASF header |

The ASF header packet size is available in `packetSize`. With this information and a packet start offset from `jumpPoints` you can parse the packet headers and skip packets in ASF files.

WMA decoder can also increase the internal clock automatically when it detects that a file can not be decoded correctly with the current clock. The maximum allowed clock is configured with the `SCI_CLOCKF` register.

10.11.3 AAC

| Parameter | Address | Usage |
|----------------|-------------|---|
| config1 | 0x1e03(7:4) | SBR and PS select |
| sceFoundMask | 0x1e2a | Single channel elements found |
| cpeFoundMask | 0x1e2b | Channel pair elements found |
| lfeFoundMask | 0x1e2c | Low frequency elements found |
| playSelect | 0x1e2d | Play element selection |
| dynCompress | 0x1e2e | Compress coefficient for DRC, -8192=1.0 |
| dynBoost | 0x1e2f | Boost coefficient for DRC, 8192=1.0 |
| sbrAndPsStatus | 0x1e30 | SBR and PS available flags |

playSelect determines which element to decode if a stream has multiple elements. The value is set to 0 each time AAC decoding starts, which causes the first element that appears in the stream to be selected for decoding. Other values are: 0x01 - select first single channel element (SCE), 0x02 - select first channel pair element (CPE), 0x03 - select first low frequency element (LFE), $S * 16 + 5$ - select SCE number S, $P * 16 + 6$ - select CPE number P, $L * 16 + 7$ - select LFE number L. When automatic selection has been performed, playSelect reflects the selected element.

sceFoundMask, cpeFoundMask, and lfeFoundMask indicate which elements have been found in an AAC stream since the variables have last been cleared. The values can be used to present an element selection menu with only the available elements.

dynCompress and dynBoost change the behavior of the dynamic range control (DRC) that is present in some AAC streams. These are also initialized when AAC decoding starts.

sbrAndPsStatus indicates spectral band replication (SBR) and parametric stereo (PS) status.

| Bit | Usage |
|-----|-------------------|
| 0 | SBR present |
| 1 | upsampling active |
| 2 | PS present |
| 3 | PS active |

Bits 7 to 4 in config1 can be used to control the SBR and PS decoding. Bits 5 and 4 select SBR mode and bits 7 and 6 select PS mode. These configuration bits are useful if your AAC license does not cover SBR and/or PS.

| config1(5:4) | Usage |
|--------------|--|
| '00' | normal mode, upsample <24 kHz AAC files |
| '01' | do not automatically upsample <24 kHz AAC files, but enable upsampling if SBR is encountered |
| '10' | never upsample |
| '11' | disable SBR (also disables PS) |

| config1(7:6) | Usage |
|--------------|--|
| '00' | normal mode, process PS if it is available |
| '01' | process PS if it is available, but in downsampled mode |
| '10' | reserved |
| '11' | disable PS processing |

AAC decoder can also increase the internal clock automatically when it detects that a file can not be decoded correctly with the current clock. The maximum allowed clock is configured with the SCI_CLOCKF register.

If even the highest allowed clock is too slow to decode an AAC file with SBR and PS components, the advanced decoding features are automatically dropped one by one until the file can be played. First the parametric stereo processing is dropped (the playback becomes mono). If that is not enough, the spectral band replication is turned into downsampled mode (reduced bandwidth). As the last resort the spectral band replication is fully disabled. Dropped features are restored at each song change.

10.11.4 Midi

| Parameter | Address | Usage |
|-----------|------------|---|
| config1 | 0x1e03 | Miscellaneous configuration |
| | bits [3:0] | Reverb: 0 = auto (ON if clock $\geq 3.0\times$) 1 = off, 2 - 15 = room size |
| bytesLeft | 0x1e2a/2b | The number of bytes left in this track |

The lowest 4 bits of config1 controls the reverb effect.

10.11.5 Ogg Vorbis

| Parameter | Address | Usage |
|-----------|---------|------------------------------|
| gain | 0x1e2a | Preferred replay-gain offset |

Ogg Vorbis decoding supports Replay Gain technology. The Replay Gain technology is used to automatically give all songs a matching volume so that the user does not need to adjust the volume setting between songs. If the Ogg Vorbis decoder finds a Replay Gain tag in the song header, the tag is parsed and the decoded gain setting can be found from the gain parameter. For a song without any Replay Gain tag, a default of -6dB (gain value -12) is used. For more details about Replay Gain, see http://en.wikipedia.org/wiki/Replay_Gain and <http://www.replaygain.org/>.

The player software can use the gain value to adjust the volume level. Negative values mean that the volume should be decreased, positive values mean that the volume should be increased.

For example $gain = -11$ means that volume should be decreased by 5.5 dB ($-11/2 = -5.5$), and left and right attenuation should be increased by 11. When $gain = 2$ volume should be increased by 1 dB ($2/2 = 1.0$), and left and right attenuation should be decreased by 2. Because volume setting can not go above +0 dB, the value should be saturated.

| Gain | Volume | SCI_VOL (Volume-Gain) |
|---------------|-------------|-----------------------|
| -11 (-5.5 dB) | 0 (+0.0 dB) | 0x0b0b (-5.5 dB) |
| -11 (-5.5 dB) | 3 (-1.5 dB) | 0x0e0e (-7.0 dB) |
| +2 (+1.0 dB) | 0 (+0.0 dB) | 0x0000 (+0.0 dB) |
| +2 (+1.0 dB) | 1 (-0.5 dB) | 0x0000 (+0.0 dB) |
| +2 (+1.0 dB) | 4 (-2.0 dB) | 0x0202 (-1.0 dB) |

10.12 SDI Tests

There are several test modes in VS1053b, which allow the user to perform memory tests, SCI bus tests, and several different sine wave tests.

All tests are started in a similar way: VS1053b is hardware reset, SM_TESTS is set, and then a test command is sent to the SDI bus. Each test is started by sending a 4-byte special command sequence, followed by 4 zeros. The sequences are described below.

10.12.1 Sine Test

Sine test is initialized with the 8-byte sequence 0x53 0xEF 0x6E *n* 0 0 0 0, where *n* defines the sine test to use. *n* is defined as follows:

| <i>n</i> bits | | |
|---------------|------|------------------|
| Name | Bits | Description |
| F_sIdx | 7:5 | Samplerate index |
| S | 4:0 | Sine skip speed |

| F_sIdx | F_s | F_sIdx | F_s |
|----------|----------|----------|----------|
| 0 | 44100 Hz | 4 | 24000 Hz |
| 1 | 48000 Hz | 5 | 16000 Hz |
| 2 | 32000 Hz | 6 | 11025 Hz |
| 3 | 22050 Hz | 7 | 12000 Hz |

The frequency of the sine to be output can now be calculated from $F = F_s \times \frac{S}{128}$.

Example: Sine test is activated with value 126, which is 0b01111110. Breaking *n* to its components, $F_sIdx = 0b011 = 3$ and thus $F_s = 22050Hz$. $S = 0b11110 = 30$, and thus the final sine frequency $F = 22050Hz \times \frac{30}{128} \approx 5168Hz$.

To exit the sine test, send the sequence 0x45 0x78 0x69 0x74 0 0 0 0.

Note: Sine test signals go through the digital volume control, so it is possible to test channels separately.

10.12.2 Pin Test

Pin test is activated with the 8-byte sequence 0x50 0xED 0x6E 0x54 0 0 0 0. This test is meant for chip production testing only.

10.12.3 SCI Test

Sci test is initialized with the 8-byte sequence 0x53 0x70 0xEE *n* 0 0 0 0, where *n* is the register number to test. The content of the given register is read and copied to SCI_HDAT0. If

the register to be tested is HDAT0, the result is copied to SCI_HDAT1.

Example: if n is 0, contents of SCI register 0 (SCI_MODE) is copied to SCI_HDAT0.

10.12.4 Memory Test

Memory test mode is initialized with the 8-byte sequence 0x4D 0xEA 0x6D 0x54 0 0 0 0. After this sequence, wait for 1100000 clock cycles. The result can be read from the SCI register SCI_HDAT0, and 'one' bits are interpreted as follows:

| Bit(s) | Mask | Meaning |
|--------|--------|--------------------|
| 15 | 0x8000 | Test finished |
| 14:10 | | Unused |
| 9 | 0x0200 | Mux test succeeded |
| 8 | 0x0100 | Good MAC RAM |
| 7 | 0x0080 | Good I RAM |
| 6 | 0x0040 | Good Y RAM |
| 5 | 0x0020 | Good X RAM |
| 4 | 0x0010 | Good I ROM 1 |
| 3 | 0x0008 | Good I ROM 2 |
| 2 | 0x0004 | Good Y ROM |
| 1 | 0x0002 | Good X ROM 1 |
| 0 | 0x0001 | Good X ROM 2 |
| | 0x83ff | All ok |

Memory tests overwrite the current contents of the RAM memories.

10.12.5 New Sine and Sweep Tests

A more frequency-accurate sine test can be started and controlled from SCI. SCI_AICTRL0 and SCI_AICTRL1 set the sine frequencies for left and right channel, respectively. These registers, volume (SCI_VOL), and samplerate (SCI_AUDATA) can be set before or during the test. Write 0x4020 to SCI_AIADDR to start the test.

SCI_AICTRL n can be calculated from the desired frequency and DAC samplerate by:

$$SCI_AICTRLn = F_{sin} \times 65536 / F_s$$

The maximum value for SCI_AICTRL n is 0x8000U. For the best S/N ratio for the generated sine, three LSB's of the SCI_AICTRL n should be zero. The resulting frequencies F_{sin} can be calculated from the DAC samplerate F_s and SCI_AICTRL0 / SCI_AICTRL1 using the following equation.

$$F_{sin} = SCI_AICTRLn \times F_s / 65536$$

Sine sweep test can be started by writing 0x4022 to SCI_AIADDR.

Both these tests use the normal audio path, thus also SCI_BASS, differential output mode, and EarSpeaker settings have an effect.

11 VS1053b Registers

11.1 Who Needs to Read This Chapter

User software is required when a user wishes to add some own functionality like DSP effects to VS1053b.

However, most users of VS1053b don't need to worry about writing their own code, or about this chapter, including those who only download software plug-ins from VLSI Solution's Web site.

Note: Also see VS1063 Hardware Guide for more information, because the hardware is compatible with VS1053.

11.2 The Processor Core

VS_DSP is a 16/32-bit DSP processor core that also had extensive all-purpose processor features. VLSI Solution's free VSKIT Software Package contains all the tools and documentation needed to write, simulate and debug Assembly Language or Extended ANSI C programs for the VS_DSP processor core. VLSI Solution also offers a full Integrated Development Environment VSIDE for full debug capabilities.

11.3 VS1053b Hardware DAC Audio Paths

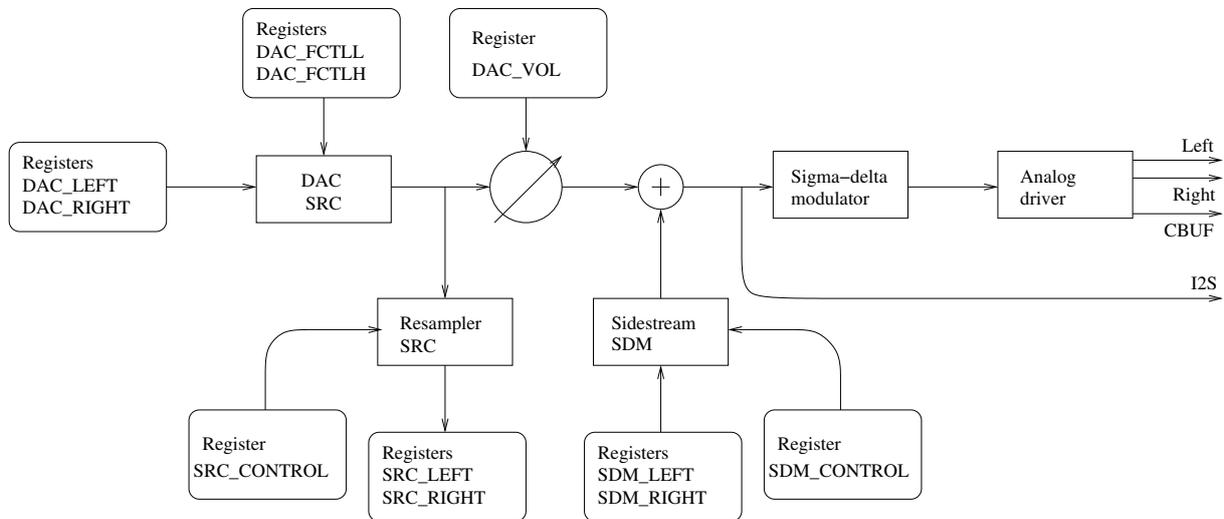


Figure 18: VS1053b ADC and DAC data paths with some data registers

Figure 18 presents the VS1053b Hardware DAC audio paths.

The main audio path starts from the DAC register (Chapter 11.8) to the high-fidelity, fully digital DAC SRC (Digital-to-Analog Converter SampleRate Converter), which low-pass filters and interpolates the data to the high samplerate of XTALI/2 (nominally 6.144 MHz). This 18-bit data is then fed to the volume control. It then passes through the sigma-delta modulator to the analog driver and analog Left and Right signals.

The user may resample and record the data with the Resampler SampleRate Converter (Chapter 11.16). Because there is no automatic low-pass filtering, it is the user's responsibility to avoid aliasing distortion.

The user may add a PCM sidestream with the Sidestream Sigma-Delta Modulator input (Chapter 11.17). As is the case with the Resampler SampleRate Converter, hardware doesn't offer low-pass filtering, so sufficient aliasing image rejection is the responsibility of the user.

11.4 VS1053b Hardware ADC Audio Paths

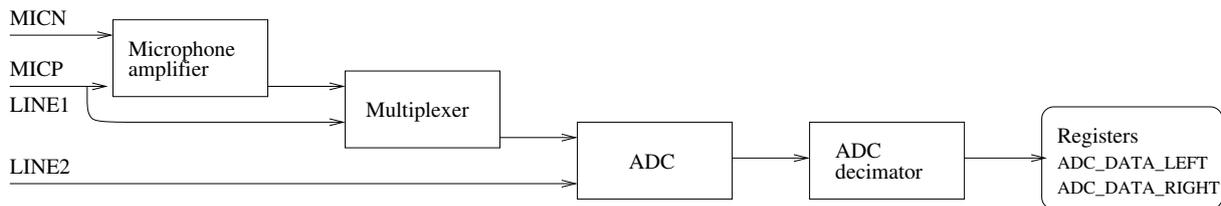


Figure 19: VS1053b ADC and DAC data paths with some data registers

Figure 18 presents the VS1053b Hardware ADC audio paths.

Analog audio may be fed upto two channels: one as a differential signal to MICN/MICP or as a one-sided signal to Line1, and the other as a one-sided signal to Line2.

If microphone input for the left channel has been selected, audio is fed through a microphone amplifier and that signal is selected by a multiplexer.

Audio is then downsampled to one of four allowed samplerrates: XTALI/64, XTALI/128, XTALI/256 or XTALI/512. With the nominal 12.288 MHz crystal, these correspond to 192, 96, 48 or 24 kHz samplerrates, respectively (Chapter 11.17).

If the “3 MHz” option bit SS_AD_CLOCK in register SCI_STATUS has been set to 1, then samplerrates are divided by two, so the nominal samplerrates become 96, 48, 24 and 12 kHz.

11.5 VS1053b Memory Map

| X-memory | | Y-memory | | I-memory | |
|----------------|-------------|----------------|-------------|----------------|-------------|
| Address | Description | Address | Description | Address | Description |
| 0x0000..0x17ff | System RAM | 0x0000..0x17ff | System RAM | 0x0000..0x004f | System RAM |
| 0x1800..0x187f | User RAM | 0x1800..0x187f | User RAM | 0x0050..0x0fff | User RAM |
| 0x1880..0x197f | Stack | 0x1880..0x197f | Stack | 0x1000..0x1fff | - |
| 0x1980..0x3fff | System RAM | 0x1980..0x3fff | System RAM | 0x2000..0xffff | ROM 56k |
| 0x4000..0xbfff | ROM 32k | 0x4000..0xdfff | ROM 40k | | and banked |
| 0xc000..0xc0ff | Peripherals | 0xe000..0xffff | System RAM | 0xc000..0xffff | ROM4 16k |
| 0xc100..0xffff | ROM 15.75k | | | | |

11.6 SCI Hardware Registers

SCI registers described in Chapter 9.6 can be found here between 0xC000..0xC00F. In addition to these registers, there is one in address 0xC010, called SCI_CHANGE.

| SCI registers, prefix SCI_ | | | | |
|----------------------------|------|-------|--------------|-------------------------|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC010 | r | 0 | CHANGE[5:0] | Last SCI access address |

| SCI_CHANGE bits | | |
|-----------------|------|------------------------------------|
| Name | Bits | Description |
| SCI_CH_WRITE | 4 | 1 if last access was a write cycle |
| SCI_CH_ADDR | 3:0 | SCI address of last access |

SCI_CHANGE contains the last SCI register that has been accessed through the SCI bus, as well as whether the access was a read or write operation.

11.7 Serial Data Interface (SDI) Registers

Whenever two bytes have been written to the SDI bus, an interrupt is generated and the data can be read as a 16-bit big-endian value from the SDI registers. The user can control the DREQ pin as if it was a general-purpose output through its own register bit.

| SDI registers, prefix SER_ | | | | |
|----------------------------|------|-------|--------------|-----------------------------------|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC011 | r | 0 | DATA | Last received 2 bytes, big-endian |
| 0xC012 | w | 0 | DREQ[0] | DREQ pin control |

11.8 DAC Registers

| DAC registers, prefix DAC_ | | | | |
|----------------------------|------|-------|--------------|--|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC013 | rw | 0 | FCTLL | DAC frequency control, 16 LSbs |
| 0xC014 | rw | 0 | FCTLH | DAC frequency control 4MSbs, PLL control |
| 0xC015 | rw | 0 | LEFT | DAC left channel PCM value |
| 0xC016 | rw | 0 | RIGHT | DAC right channel PCM value |
| 0xC045 | rw | 0 | VOL | DAC hardware volume |

The internal 20-bit register DAC_FCTL is calculated from DAC_FCTLH and DAC_FCTLL registers as follows: $DAC_FCTL = (DAC_FCTLH \& 15) \times 65536 + DAC_FCTLL$. Highest supported value for DAC_FCTL is 0x80000.

If we define $C = DAC_FCTL$ and $X = XTALI$ in Hz, then the resulting samplerate f_s of the associated DAC SampleRate Converter is $f_s = C \times X \times 2^{-27}$.

Example:

If $C = 0x80000$ and $X = 12.288$ MHz then $f_s = 524288 \times (12.288 \times 10^6) \times 2^{-27} = 48000$ (Hz).

Note: FCTLH bits 13:4 are used for the PLL Controller. See Chapter 11.9 for details.

| DAC_VOL bits | | |
|--------------|-------|--|
| Name | Bits | Description |
| LEFT_FINE | 15:12 | Left channel gain +0.0 dB...+5.5 dB (0 to 11) |
| LEFT_COARSE | 11:8 | Left channel attenuation in -6 dB steps |
| RIGHT_FINE | 7:4 | Right channel volume +0.0 dB...+5.5 dB (0 to 11) |
| RIGHT_COARSE | 3:0 | Right channel attenuation in -6 dB steps |

Normally DAC_VOL is handled by the firmware. DAC_VOL depends on SCI_VOL and the bass and treble settings in SCI_BASS (and optionally SS_SWING bits in SCI_STATUS).

11.9 PLL Controller

The Phase-Locked Loop (PLL) controller is used to generate clock frequencies that are higher than the incoming (crystal-based) clock frequency. The PLL output is used by the CPU core and some peripherals.

Configurable features include:

- VCO Enable/Disable
- Select VCO or input clock to be output clock
- Route VCO frequency to output pin

- Select PLL clock multiplier

At the core of the PLL controller is the VCO, a high frequency oscillator, whose oscillation frequency is adjusted to be an integer multiple of some input frequency. As the name “Phase-Locked Loop” suggests, this is done by comparing the phase of the input frequency against the phase of a signal which is derived from the VCO output through frequency division.

If the system is stable, e.g. the comparison phase difference remains virtually zero, the PLL is said to be “in lock”. This means that the output frequency of the VCO is stable and reliable.

The PLL is preceded by a division-by-two unit. Thus, with a nominal XTALI = 12.288 MHz, the internal clock frequency CLKI can be adjusted with an accuracy of XTALI/2 = 6.144 MHz.

PLL control lies in DAC_FCTL bits 13:4. To see what bits 3:0 do, see Chapter 11.8.

| FREQCTLH PLL bits, prefix FCH_ | | |
|--------------------------------|------|---|
| Name | Bits | Description |
| PLL_LOCKED | 13 | 0=lock failed since last test (read-only) |
| PLL_SET_LOCK | 12 | 1:Sets FCH_PLL_LOCKED to 1 to start lock test |
| PLL_VCO_OUT_ENA | 11 | Route VCO to GPIO pin (VS1000:second cs pin) |
| PLL_FORCE_PLL | 9 | 1:System clock is VCO / 0:System clock is inclk |
| PLL_DIV_INCLK | 8 | divide inclk by 2 (for 1.5, 2.5 or 3.5 x clk) |
| PLL_RATE | 7:4 | PLL rate control |

The PLL locked status can be checked by generating a high-active pulse (writing first “1”, then “0”) to FCH_PLL_SET_LOCK and reading FCH_PLL_LOCKED. FCH_PLL_LOCKED is set to “1” along with the high level of FCH_PLL_SET_LOCK and to “0” whenever the PLL falls out of lock. So if the “1” remains in FCH_PLL_LOCKED, PLL is in sync.

The PLL controller’s operation is optimized for frequencies around 12. . . 13 MHz. If you use an 24. . . 26 MHz input clock, set the extra clock divider bit SM_CLK_RANGE in register SCI_MODE to 1 before activating the PLL.

It’s recommended to change the PLL rate in small steps and wait for the PLL to stabilize after each change. For diagnostic purposes, the PLL clock output (VCO) can be routed to an I/O pin so it can be scanned with an oscilloscope.

FCH_PLL_RATE (bits 7:4) control PLL multiplication rate. PLL multiplier is (FCH_PLL_RATE + 1). When FCH_PLL_RATE is 0, the VCO is powered down and output clock is forced to be input clock (same as if FCH_PLL_FORCE_PLL = 0).

11.10 GPIO

| GPIO registers, prefix GPIO_ | | | | |
|------------------------------|------|-------|--------------|---------------------------|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC017 | rw | 0 | DDR[7:0] | Direction |
| 0xC018 | r | 0 | IDATA[11:0] | Values read from the pins |
| 0xC019 | rw | 0 | ODATA[7:0] | Values set to the pins |

GPIO_DIR is used to set the direction of the GPIO pins. 1 means output. GPIO_ODATA remembers its values even if a GPIO_DIR bit is set to input.

GPIO_IDATA is used to read the pin states. In VS1053 also the SDI and SCI input pins can be read through GPIO_IDATA: SCLK = GPIO_IDATA[8], XCS = GPIO_IDATA[9], SI = GPIO_IDATA[10], and XDCS = GPIO_IDATA[11].

GPIO registers don't generate interrupts.

Note that in VS1053b the VSDSP registers can be read and written through the SCI_WRAMADDR and SCI_WRAM registers. You can thus use the GPIO pins quite conveniently.

11.11 Interrupt Control

| Interrupt registers, prefix INT_ | | | | |
|----------------------------------|------|-------|--------------|--|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC01A | rw | 0 | ENABLE[9:0] | Interrupt enable |
| 0xC01B | w | 0 | GLOB_DIS[-] | Write to add to interrupt counter |
| 0xC01C | w | 0 | GLOB_ENA[-] | Write to subtract from interrupt counter |
| 0xC01D | rw | 0 | COUNTER[4:0] | Interrupt counter |

INT_ENABLE controls the interrupts. The control bits are as follows:

| INT_ENABLE bits | | |
|-----------------|------|--|
| Name | Bits | Description |
| INT_EN_SDM | 9 | Enable Sigma Delta Modulator interrupt |
| INT_EN_SRC | 8 | Enable SampleRate Converter interrupt |
| INT_EN_TIM1 | 7 | Enable Timer 1 interrupt |
| INT_EN_TIM0 | 6 | Enable Timer 0 interrupt |
| INT_EN_RX | 5 | Enable UART RX interrupt |
| INT_EN_TX | 4 | Enable UART TX interrupt |
| INT_EN_ADC | 3 | Enable AD modulator interrupt |
| INT_EN_SDI | 2 | Enable Data interrupt |
| INT_EN_SCI | 1 | Enable SCI interrupt |
| INT_EN_DAC | 0 | Enable DAC interrupt |

Note: It may take upto 6 clock cycles before changing INT_ENABLE has any effect.

Writing any value to INT_GLOB_DIS adds one to the interrupt counter INT_COUNTER and effectively disables all interrupts. It may take upto 6 clock cycles before writing to this register has any effect.

Writing any value to INT_GLOB_ENA subtracts one from the interrupt counter INT_COUNTER, unless it already was 0, in which case nothing happens. If, after the operation INT_COUNTER becomes zero, interrupts selected with INT_ENABLE are restored. An interrupt routine should always write to this register as the last thing it does, because interrupts automatically add one to the interrupt counter, but subtracting it back to its initial value is the responsibility of the user. It may take upto 6 clock cycles before writing this register has any effect.

By reading INT_COUNTER the user may check if the interrupt counter is correct or not. If the register is not 0, interrupts are disabled.

11.12 UART

RS232 UART implements a serial interface using rs232 standard.



Figure 20: RS232 serial interface protocol

When the line is idling, it stays in logic high state. When a byte is transmitted, the transmission begins with a start bit (logic zero) and continues with data bits (LSB first) and ends up with a stop bit (logic high). 10 bits are sent for each 8-bit byte frame.

11.12.1 UART Registers

| UART registers, prefix UART_ | | | | |
|------------------------------|------|-------|-------------|-------------|
| Reg | Type | Reset | Abbrev | Description |
| 0xC028 | r | 0 | STATUS[4:0] | Status |
| 0xC029 | r/w | 0 | DATA[7:0] | Data |
| 0xC02A | r/w | 0 | DATAH[15:8] | Data High |
| 0xC02B | r/w | 0 | DIV | Divider |

11.12.2 Status UART_STATUS

A read from the status register returns the transmitter and receiver states.

| UART_STATUS Bits | | |
|-------------------|------|--------------------------------|
| Name | Bits | Description |
| UART_ST_FRAMEERR | 4 | Framing error (stop bit was 0) |
| UART_ST_RXORUN | 3 | Receiver overrun |
| UART_ST_RXFULL | 2 | Receiver data register full |
| UART_ST_TXFULL | 1 | Transmitter data register full |
| UART_ST_TXRUNNING | 0 | Transmitter running |

UART_ST_FRAMEERR is set if the stop bit of the received byte was 0.

UART_ST_RXORUN is set if a received byte overwrites unread data when it is transferred from the receiver shift register to the data register, otherwise it is cleared.

UART_ST_RXFULL is set if there is unread data in the data register.

UART_ST_TXFULL is set if a write to the data register is not allowed (data register full).

UART_ST_TXRUNNING is set if the transmitter shift register is in operation.

11.12.3 Data UART_DATA

A read from UART_DATA returns the received byte in bits 7:0, bits 15:8 are returned as '0'. If there is no more data to be read, the receiver data register full indicator will be cleared.

A receive interrupt will be generated when a byte is moved from the receiver shift register to the receiver data register.

A write to UART_DATA sets a byte for transmission. The data is taken from bits 7:0, other bits in the written value are ignored. If the transmitter is idle, the byte is immediately moved to the transmitter shift register, a transmit interrupt request is generated, and transmission is started. If the transmitter is busy, the UART_ST_TXFULL will be set and the byte remains in the transmitter data register until the previous byte has been sent and transmission can proceed.

11.12.4 Data High UART_DATAH

The same as UART_DATA, except that bits 15:8 are used.

11.12.5 Divider UART_DIV

| UART_DIV Bits | | |
|---------------|------|--------------------|
| Name | Bits | Description |
| UART_DIV_D1 | 15:8 | Divider 1 (0..255) |
| UART_DIV_D2 | 7:0 | Divider 2 (6..255) |

The divider is set to 0x0000 in reset. The ROM boot code must initialize it correctly depending on the master clock frequency to get the correct bit speed. The second divider (D_2) must be from 6 to 255.

The communication speed $f = \frac{f_m}{(D_1+1) \times (D_2)}$, where f_m is the master clock frequency, and f is the TX/RX speed in bps.

Divider values for common communication speeds at 26 MHz master clock:

| Example UART Speeds, $f_m = 49.152 MHz$ | | |
|---|-------------|-------------|
| Comm. Speed [bps] | UART_DIV_D1 | UART_DIV_D2 |
| 4800 | 255 | 40 |
| 9600 | 255 | 20 |
| 14400 | 233 | 15 |
| 19200 | 255 | 10 |
| 28800 | 243 | 7 |
| 38400 | 159 | 8 |
| 57600 | 121 | 7 |
| 115200 | 60 | 7 |

11.12.6 UART Interrupts and Operation

Transmitter operates as follows: After an 8-bit word is written to the transmit data register it will be transmitted instantly if the transmitter is not busy transmitting the previous byte. When the transmission begins a TX_INTR interrupt will be sent. Status bit [1] informs the transmitter data register empty (or full state) and bit [0] informs the transmitter (shift register) empty state. A new word must not be written to transmitter data register if it is not empty (bit [1] = '0'). The transmitter data register will be empty as soon as it is shifted to transmitter and the transmission is begun. It is safe to write a new word to transmitter data register every time a transmit interrupt is generated.

Receiver operates as follows: It samples the RX signal line and if it detects a high to low transition, a start bit is found. After this it samples each 8 bit at the middle of the bit time (using a constant timer), and fills the receiver (shift register) LSB first. Finally the data in the receiver is moved to the receive data register, the stop bit state is checked (logic high = ok, logic low = framing error) for status bit[4], the RX_INTR interrupt is sent, status bit[2] (receive data register full) is set, and status bit[2] old state is copied to bit[3] (receive data overrun). After that the receiver returns to idle state to wait for a new start bit. Status bit[2] is zeroed when the receiver data register is read.

RS232 communication speed is set using two clock dividers. The base clock is the processor master clock. Bits 15-8 in these registers are for first divider and bits 7-0 for second divider. RX sample frequency is the clock frequency that is input for the second divider.

11.13 Timers

There are two 32-bit timers that can be initialized and enabled independently of each other. If enabled, a timer initializes to its start value, written by a processor, and starts decrementing every clock cycle. When the value goes past zero, an interrupt is sent, and the timer initializes to the value in its start value register, and continues downcounting. A timer stays in that loop as long as it is enabled.

A timer has a 32-bit timer register for down counting and a 32-bit TIMER1_LH register for holding the timer start value written by the processor. Timers have also a 2-bit TIMER_ENA register. Each timer is enabled (1) or disabled (0) by a corresponding bit of the enable register.

11.13.1 Timer Registers

| Timer registers, prefix TIMER_ | | | | |
|--------------------------------|------|-------|-------------|--------------------------|
| Reg | Type | Reset | Abbrev | Description |
| 0xC030 | r/w | 0 | CONFIG[7:0] | Timer configuration |
| 0xC031 | r/w | 0 | ENABLE[1:0] | Timer enable |
| 0xC034 | r/w | 0 | T0L | Timer0 startvalue - LSBs |
| 0xC035 | r/w | 0 | T0H | Timer0 startvalue - MSBs |
| 0xC036 | r/w | 0 | T0CNTL | Timer0 counter - LSBs |
| 0xC037 | r/w | 0 | T0CNTH | Timer0 counter - MSBs |
| 0xC038 | r/w | 0 | T1L | Timer1 startvalue - LSBs |
| 0xC039 | r/w | 0 | T1H | Timer1 startvalue - MSBs |
| 0xC03A | r/w | 0 | T1CNTL | Timer1 counter - LSBs |
| 0xC03B | r/w | 0 | T1CNTH | Timer1 counter - MSBs |

11.13.2 Configuration TIMER_CONFIG

| TIMER_CONFIG Bits | | |
|-------------------|------|----------------------|
| Name | Bits | Description |
| TIMER_CF_CLKDIV | 7:0 | Master clock divider |

TIMER_CF_CLKDIV is the master clock divider for all timer clocks. The generated internal clock frequency $f_i = \frac{f_m}{c+1}$, where f_m is the master clock frequency and c is TIMER_CF_CLKDIV. Example: With a 12 MHz master clock, TIMER_CF_DIV=3 divides the master clock by 4, and the output/sampling clock would thus be $f_i = \frac{12MHz}{3+1} = 3MHz$.

11.13.3 Configuration **TIMER_ENABLE**

| TIMER_ENABLE Bits | | |
|-------------------|------|----------------|
| Name | Bits | Description |
| TIMER_EN_T1 | 1 | Enable timer 1 |
| TIMER_EN_T0 | 0 | Enable timer 0 |

11.13.4 Timer X Startvalue **TIMER_Tx[L/H]**

The 32-bit start value **TIMER_Tx[L/H]** sets the initial counter value when the timer is reset. The timer interrupt frequency $f_t = \frac{f_i}{c+1}$ where f_i is the master clock obtained with the clock divider (see Chapter 11.13.2 and c is **TIMER_Tx[L/H]**).

Example: With a 12 MHz master clock and with **TIMER_CF_CLKDIV**=3, the master clock $f_i = 3MHz$. If **TIMER_TH**=0, **TIMER_TL**=99, then the timer interrupt frequency $f_t = \frac{3MHz}{99+1} = 30kHz$.

11.13.5 Timer X Counter **TIMER_TxCNT[L/H]**

TIMER_TxCNT[L/H] contains the current counter values. By reading this register pair, the user may get knowledge of how long it will take before the next timer interrupt. Also, by writing to this register, a one-shot different length timer interrupt delay may be realized.

11.13.6 Timer Interrupts

Each timer has its own interrupt, which is asserted when the timer counter underflows.

11.14 I2S DAC Interface

The I2S Interface makes it possible to attach an external DAC to the system.

Note: The samplerate of the audio file and the I2S rate are independent. All audio will be automatically converted to 6.144 MHz for VS1053 DAC and to the configured I2S rate using a high-quality sample-rate converter.

Note: In VS1053b the I2S pins share different GPIO pins than in VS1033 to be able to use SPI boot and I2S in the same application.

| I2S registers, prefix I2S_ | | | | |
|----------------------------|------|-------|-------------|-------------------|
| Reg | Type | Reset | Abbrev | Description |
| 0xC040 | r/w | 0 | CONFIG[3:0] | I2S configuration |

| I2S_CONFIG Bits | | |
|-----------------|------|--|
| Name | Bits | Description |
| I2S_CF_MCLK_ENA | 3 | Enables the MCLK output (12.288 MHz) |
| I2S_CF_ENA | 2 | Enables I2S, otherwise pins are GPIO |
| I2S_CF_SRATE | 1:0 | I2S rate, "10" = 192, "01" = 96, "00" = 48 kHz |

I2S_CF_ENA enables the I2S interface. After reset I2S is disabled and the pins are used for GPIO inputs.

I2S_CF_MCLK_ENA enables the MCLK output. The frequency is either directly the input clock (nominal 12.288 MHz), or half the input clock when mode register bit SM_CLK_RANGE is set to 1 (24-26 MHz input clock).

I2S_CF_SRATE controls the output samplerate. When set to 48 kHz, SCLK is MCLK divided by 8, when 96 kHz SCLK is MCLK divided by 4, and when 192 kHz SCLK is MCLK divided by 2. I2S_CF_SRATE can only be changed when I2S_CF_ENA is 0.

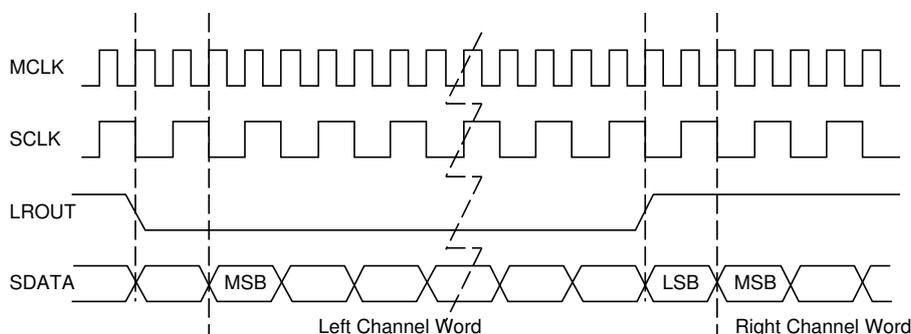


Figure 21: I2S interface, 192 kHz.

To enable I2S first write 0xc017 to SCI_WRAMADDR and 0xf0 to SCI_WRAM, then write 0xc040 to SCI_WRAMADDR and 0x0c to SCI_WRAM.

11.15 Analog-to-Digital Converter (ADC)

ADC modulator registers control Analog-to-Digital conversions of VS1053b.

| ADC Decimator registers, prefix ADC_ | | | | |
|--------------------------------------|------|-------|--------------|------------------------|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC042 | rw | 0 | CONTROL[4:0] | ADC control |
| 0xC043 | r | 0 | DATA_LEFT | ADC left channel data |
| 0xC044 | r | 0 | DATA_RIGHT | ADC right channel data |

ADC_CONTROL controls the ADC and its associated decimator unit.

| ADC_CONTROL Bits | | |
|------------------|------|---|
| Name | Bits | Description |
| ADC_MODU2_PD | 4 | Right channel powerdown |
| ADC_MODU1_PD | 3 | Left channel powerdown |
| ADC_DECIM_FACTOR | 2:1 | ADC Decimator factor: - 3 = downsample to XTALI/512 (nominal 24 kHz) - 2 = downsample to XTALI/256 (nominal 48 kHz) - 1 = downsample to XTALI/128 (nominal 96 kHz) - 0 = downsample to XTALI/64 (nominal 192 kHz) |
| ADC_ENABLE | 0 | Set to activate ADC converter and decimator |

Note: Setting bit SS_AD_CLOCK in register SCI_STATUS will halve the operation speed of the A/D unit, and thus halve the resulting samplerate.

Each time a new (stereo) sample has been generated, an ADC interrupt is generated.

11.16 Resampler SampleRate Converter (SRC)

The resampler SRC makes it possible to catch audio from the DAC path.

Note: hardware makes no attempts at low-pass filtering data. If the SRC samplerate is lower than the DAC samplerate, aliasing may and will occur.

| Resampler SRC registers, prefix SRC_ | | | | |
|--------------------------------------|------|-------|---------------|------------------------|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC046 | rw | 0 | CONTROL[12:0] | SRC control |
| 0xC047 | r | 0 | DATA_LEFT | SRC left channel data |
| 0xC048 | r | 0 | DATA_RIGHT | SRC right channel data |

| SRC_CONTROL Bits | | |
|------------------|------|---------------------------------------|
| Name | Bits | Description |
| SRC_ENABLE | 12 | Set to enable SRC |
| SRC_DIV | 11:0 | Set samplerate to XTALI/2/(SRC_DIV+1) |

Each time a new (stereo) sample has been generated, an SRC interrupt is generated.

11.17 Sidestream Sigma-Delta Modulator (SDM)

The Sidestream Sigma-Delta Modulator makes it possible to insert a digital side stream on top of existing audio.

Note: The SDM provides a direct, low-delay side channel to the Sigma-Delta DACs of VS10xx. It makes no attempts at low-pass filtering data. Thus there will be practically no image rejection. If using low samplerrates, this may cause audible aliasing distortion.

| Sidestream SDM registers, prefix SDM_ | | | | |
|---------------------------------------|------|-------|---------------|------------------------|
| Reg | Type | Reset | Abbrev[bits] | Description |
| 0xC049 | rw | 0 | CONTROL[12:0] | SDM control |
| 0xC04A | rw | 0 | DATA_LEFT | SDM left channel data |
| 0xC04B | rw | 0 | DATA_RIGHT | SDM right channel data |

| SDM_CONTROL Bits | | |
|------------------|------|---------------------------------------|
| Name | Bits | Description |
| SDM_ENABLE | 12 | Set to enable SDM |
| SDM_DIV | 11:0 | Set samplerate to XTALI/2/(SDM_DIV+1) |

Each time a new (stereo) sample is needed, an SDM interrupt is generated.

12 Version Changes

This chapter describes the latest and most important changes done to VS1053b

12.1 Changes Between VS1033c and VS1053a/b Firmware, 2007-03-08

Completely new or major changes:

- I2S pins are now in GPIO4-GPIO7 and do not overlap with SPI boot pins.
- No software reset required between files when used correctly.
- Ogg Vorbis decoding added. Non-fatal ogg or vorbis decode errors cause automatic resync. This allows easy rewind and fast forward. Decoding ends if the "last frame" flag is reached or SM_CANCEL is set.
- HE-AAC v2 Level 3 decoding added. It is possible to disable PS and SBR processing and control the upsampling modes through `parametric_x.control1`.
- Like the WMA decoder, the AAC decoder uses the clock adder (see SCI_CLOCKF) if it needs more clock to decode the file. HE-AAC features are dropped one by one, if the file can not be decoded correctly even with the highest allowed clock. Parametric stereo is the first feature to be dropped, then downsampled mode is used, and as the final resort Spectral Band Replication is disabled. Features are automatically restored for the next file.
- Completely new volume control with zero-cross detection prevents pops when volume is changed.
- Audio FIFO underrun detection (with slow fade to zero) instead of looping the audio buffer content.
- Average bitrate calculation (`byteRate`) for all codecs.
- All codecs support fast play mode with selectable speeds for the best-quality fast forward operation. Fast play also advances DECODE_TIME faster.
- WMA and Ogg Vorbis provide an absolute decode position in milliseconds.
- When SM_CANCEL is detected, the firmware also discards the stream buffer contents.
- Bit SCIST_DO_NOT_JUMP in SCI_STATUS is '1' when jumps in the file should not be done: during header processing and with Midi files.
- IMA ADPCM encode now supports stereo encoding and selectable samplerate.

Other changes or additions:

- Delayed volume and bass/treble control calculation reduces the time the corresponding SCI operations take. This delayed handling and the new volume control hardware prevents audio samples from being missed during volume change.
- SCI_DECODE_TIME only cleared at hardware and software reset to allow files to be played back-to-back or looped.

- Read and write to YRAM at 0xe000..0xffff added to SCI_WRAMADDR/SCI_WRAM.
- The `resync` parameter (`parametric_x.resync`) is set to 32767 after reset to allow infinite resynchronization attempts (or until `SM_CANCEL` is set). Old operation can be restored by writing 0 to `resync` after reset.
- WMA,AAC: more robust resync.
- WMA,AAC: If resync is performed, broadcast mode is automatically activated. The broadcast mode disables file size checking, and decoding continues until `SM_CANCEL` is set or reset is performed.
- Treble control fixed (volume change could cause bad artefacts).
- MPEG Layer I mono fixed.
- MPEG Layer II half-rate decoding fixed (frame size was calculated wrong).
- MPEG Layer II accuracy problem fixed, invalid grouped values set to 0.
- WAV parser now skips unknown RIFF chunks.
- IMA ADPCM: Maximum blocksize is now 4096 bytes (4088 samples stereo, 8184 mono). Thus, now also plays 44100Hz stereo.
- Rt-midi: starts if in reset `GPIO0='0'`, `GPIO1='1'`, `GPIO2&3` give earSpeaker setup.
- `NewSinTest()` and `NewSinSweep()` added (`AIADDR = 0x4020/0x4022`) `AICTRL0` and `AICTRL1` set sin frequency for left/right.
- Clears memory before SPI boot and not in `InitHardware()`.

Known quirks, bugs, or features in VS1053b:

- Setting volume clears `SS_REFERENCE_SEL` and `SS_AD_CLOCK` bits. See Chapter 9.6.2.
- Software reset clears `GPIO_DDR`, also affects I2S pins.
- Ogg Vorbis occasionally overflows in windowing causing a small glitch to audio. Patch available (*VS1053b Patches w/ FLAC Decoder* plugin at <http://www.vlsi.fi/en/support/software/vs10xxplugins.html>).
- IMA ADPCM encoding requires short patch to start. Patch available in Chapter 10.8.1.
- There are also fixes for some other issues, we recommend you use the latest version of the *VS1053b Patches w/ FLAC Decoder* package from <http://www.vlsi.fi/en/support/software/vs10xxplugins.html>.

13 Document Version Changes

This chapter describes the most important changes to this document.

Version 1.20, 2012-12-03

- Major update to Chapter 11 *VS1053b Registers*. Added Chapter 11.3, *VS1053b Hardware DAC Audio Paths*, Chapter 11.4, *VS1053b Hardware ADC Audio Paths*, Chapter 11.9, *PLL Controller*, Chapter 11.15, *Analog-to-Digital Converter (ADC)*, Chapter 11.16, *Resampler SampleRate Converter (SRC)*, and Chapter 11.17, *Sidestream Sigma-Delta Modulator (SDM)*. Also revised several other sections.
- Fixed SCI_MODE default value in Chapter 9.6.1, *SCI_MODE (RW)*.
- Added info on how to read DREQ through SCI to Chapter 7.2, *Data Request Pin DREQ*.
- Slight reorganization to make datasheet more similar to VS1063a Datasheet.

Version 1.13, 2011-05-27

- xRESET, XTALI and XTALO high-level are referenced from IOVDD in Chapter 4.5.

Version 1.12, 2010-10-28

- Fixed the real-time MIDI through SDI documentation.

Version 1.11, 2010-04-30

- Minor updates.

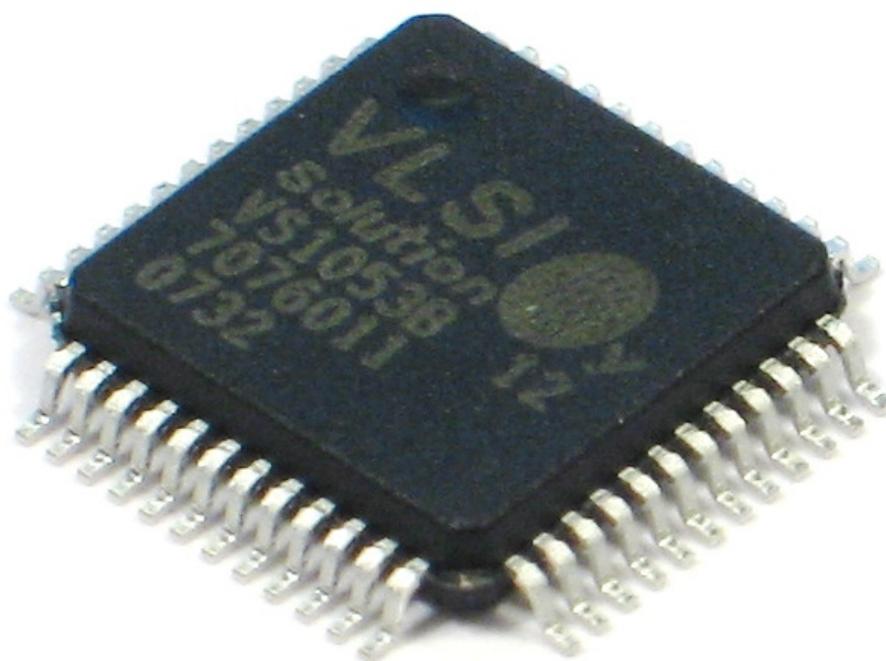
Version 1.10, 2009-09-04

- Added mentions of new Ogg Vorbis encoder and FLAC decoder plugins.
- PCM recording documentation enhanced (Chapters 10.8 and 10.8.4).
- SCLK, XCS, SI, XDACS can be read through GPIO_IDATA.
- I2S rate and audio rate are independent.

14 Contact Information

VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

Fax: +358-3-3140-8288
Phone: +358-3-3140-8200
Email: sales@vlsi.fi
URL: <http://www.vlsi.fi/>



VS1053B OGG VORBIS ENCODER

VSMPEG "VLSI Solution Audio Decoder"

Project Code:
Project Name: VSMPEG

All information in this document is provided as-is without warranty. Features are subject to change without notice.

| Revision History | | | |
|-------------------------|-------------|---------------|--|
| Rev. | Date | Author | Description |
| 1.70c | 2012-11-16 | HH | Enhanced streaming documentation. |
| 1.70b | 2011-06-29 | HH | Added .IMG files and example C source code. |
| 1.70 | 2010-09-09 | HH | Added Vox and Pause control for mono profiles. |
| 1.60d | 2010-08-10 | HH | Corrected Figure 3 on page 15. |
| 1.60c | 2010-05-10 | HH | Enhanced this document, removed XP/Vista demo. |
| 1.60b | 2010-05-04 | HH | Corrected formatting error in this document. |
| 1.60 | 2010-03-30 | HH | Enhanced sound quality, added stereo VU meter. |
| 1.50 | 2009-07-02 | HH | Added byte alignment, mono channel selection. |
| 1.40 | 2009-03-24 | HH | SCI_CLOCKF bug fix. |
| 1.31 | 2009-02-26 | HH | Added Chapter 5.3. |
| 1.30 | 2009-01-15 | HH | Added sample counter. |
| 1.21 | 2008-11-21 | HH | Corrected contents for SCI_CLOCKF. |
| 1.20 | 2008-11-20 | HH | Many new features, see Chapter 6. |
| 1.10 | 2008-05-22 | HH | Improved quality, many new profiles. |
| 1.01 | 2007-12-21 | HH | Added XP/Vista demo binary. |
| 1.00 | 2007-12-03 | HH | Initial version. |

Contents

| | |
|--|-----------|
| VS1053b Ogg Vorbis Encoder Front Page | 1 |
| Table of Contents | 2 |
| 1 Introduction | 4 |
| 2 Using the Ogg Vorbis Encoder | 5 |
| 2.1 Limitations and Requirements | 5 |
| 2.2 Ogg Vorbis Encoder Profiles | 5 |
| 2.2.1 Profile Groups | 5 |
| 2.2.2 Profile Quality Settings | 6 |
| 2.3 Running the VS1053b Ogg Vorbis Encoder | 8 |
| 2.3.1 VS1053b Ogg Vorbis Encoder Registers | 8 |
| 2.3.2 Loading and Starting the Code | 9 |
| 2.3.3 Reading Ogg Vorbis Data | 10 |
| 2.3.4 Reading Additional Data while Recording Ogg Vorbis | 11 |
| 2.3.5 Finishing Ogg Vorbis Recording | 11 |
| 2.4 Recording Levels and Automatic Gain Control (AGC) | 12 |
| 2.4.1 Reading the Recording Level | 12 |
| 2.4.2 Setting AGC | 13 |
| 2.4.3 Building a Useful VU Meter | 14 |
| 2.4.4 Converting from Linear to Decibel Scale | 16 |
| 2.4.5 vs1053oggrec.c | 17 |
| 2.5 Vox and Pause Operation | 22 |
| 2.5.1 Vox and Pause Registers | 23 |
| 2.5.2 Vox Output to GPIO4 | 23 |
| 2.6 Samplerate Considerations | 24 |
| 2.7 Post-Processing the Recording with VorbisGain | 24 |
| 3 The Ogg Vorbis Format | 25 |
| 3.1 Introduction to Ogg Vorbis | 25 |
| 3.1.1 Variable Bit-Rate | 25 |
| 3.1.2 Transcoding | 26 |
| 3.2 Compressing into Ogg Vorbis with a PC | 27 |
| 3.2.1 Preparation | 27 |
| 3.2.2 Compressing CD-Quality Music | 27 |
| 3.2.3 Compressing Speech | 29 |
| 3.3 Post-Processing with VorbisGain | 30 |
| 4 How to Load a Plugin | 31 |
| 4.1 How to Load a .PLG File | 31 |
| 4.2 How to Load an .IMG File | 32 |
| 4.2.1 Example Microcontroller .IMG File Decoder | 33 |
| 5 Building a Streaming System | 34 |
| 5.1 Sample Counter | 35 |
| 5.2 End-to-End Delay of a Streaming System | 36 |

| | | |
|----------|-------------------------------------|-----------|
| 5.3 | Playback Speed Adjustment | 36 |
| 5.4 | Streaming Example | 37 |
| 6 | Latest Version Changes | 38 |
| 7 | Contact Information | 39 |

List of Figures

| | | |
|---|---|----|
| 1 | Example colour VU meter. | 14 |
| 2 | Example monochrome VU meter. | 14 |
| 3 | Real record display with monochrome VU meter. | 15 |
| 4 | Vox / Pause logic. | 22 |
| 5 | Streaming system using VS1053b. | 34 |

1 Introduction

The VS1053b allows the user to encode files into the highly efficient Ogg Vorbis format. This makes it possible for the user to build a device that can record high-quality stereo sound while retaining moderate file sizes.

This document is an instruction manual on how to use the VS1053b Ogg Vorbis encoder application as well as an introduction to the Ogg Vorbis format itself. Example code in C is also provided.

Chapter 2 describes how to load and run different Ogg Vorbis profiles on a VS1053b. It also discusses recording levels and recording level meters.

Chapter 3 briefly introduces the Ogg Vorbis format. It also has some suggestions on how to encode data to Ogg Vorbis format as efficiently as possible on a PC. If you are unfamiliar with the Ogg Vorbis format, please read this chapter first.

Chapter 4 tells how to load a plugin code to a VS10XX chip.

Chapter 5 describes how a streaming VS1053b system can be built.

The document version history for the latest releases is provided in Chapter 6.

Finally, Chapter 7 contains VLSI Solution's contact information.

2 Using the Ogg Vorbis Encoder

The VS1053b Ogg Vorbis Encoder application is provided as a plugin in the same package as this document, downloadable at <http://www.vlsi.fi/en/support/software/vs10xxapplications.html>.

Before loading the application into your VS1053b, select one of the profiles shown below.

2.1 Limitations and Requirements

- Maximum SPI (SCI) clock speed is 3.5 Mbit/s. If a higher speed is used, there may be occasional data read errors.

2.2 Ogg Vorbis Encoder Profiles

There are 45 different Ogg Vorbis profiles available. They are divided into five groups, depending on their samplerate and number of channels.

2.2.1 Profile Groups

The five profile groups are presented below. As a comparison, bitrate figures for uncompressed 16-bit PCM (WAV) files with similar specifications are also shown.

| Ogg Vorbis Profiles | | | | | | | |
|-----------------------|------------------------|----------|----|--------------------------|-------------------------|-----------|-------------------------|
| Profile name | File name ¹ | SRate Hz | Ch | BRat ² kbit/s | WAV ³ kbit/s | Time h/GB | Vox+ Pause ⁴ |
| Voice | venc08k1qXX.plg | 8000 | 1 | 15 | 128 | 149 | Y |
| Wideband Voice | venc16k1qXX.plg | 16000 | 1 | 28 | 256 | 79 | Y |
| Wideband Stereo Voice | venc16k2qXX.plg | 16000 | 2 | 49 | 512 | 45 | N |
| HiFi Voice | venc44k1qXX.plg | 44100 | 1 | 87 | 706 | 26 | Y |
| Stereo Music | venc44k2qXX.plg | 44100 | 2 | 135 | 1411 | 16 | N |

¹ Replace XX with a quality value between 00 and 10. See Chapter 2.2.2 for details.

² Estimate for quality value 05. Actual bitrate depends on content being recorded.

³ Comparison bitrate for a 16-bit WAV file with the same specifications.

⁴ For details on Vox and Pause operation, see Chapter 2.5.

2.2.2 Profile Quality Settings

Each of the low-sample-rate profile groups has eleven quality profiles, numbered from 00 to 10. The high-sample-rate profile groups have six quality profiles from 00 to 05. This chapter presents the five profile groups and gives estimates of typical bitrates that can be obtained with the profile.

In general, quality setting 05 is designed to be a typical value that gives good quality for the application. Qualities 00 and 01 are for emergency use only when there just isn't storage space for better quality settings. High qualities, starting from 07 or 08, are intended for transparent sound quality, the intent being that compression cannot be heard anymore even under stringent circumstances.

The "Voice" Profiles

The "Voice" profiles are intended for speech applications.

These profiles support Vox and Pause.

| Voice | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|-----------|----|----|----|----|----|
| Profile number | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 |
| Typical kbit/s | 6 | 8 | 10 | 11 | 13 | 15 | 20 | 24 | 27 | 30 | 33 |

The "Wideband Voice" Profiles

"Wideband Voice" is intended to be used when high speech quality is required.

These profiles support Vox and Pause.

| Wideband Voice | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|-----------|----|----|----|----|----|
| Profile number | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 |
| Typical kbit/s | 6 | 10 | 14 | 18 | 23 | 28 | 37 | 45 | 52 | 58 | 63 |

The "Wideband Stereo Voice" Profiles

"Wideband Stereo Voice" is intended to be used when high speech quality with directional information is required.

These profiles don't support Vox and Pause.

| Wideband Stereo Voice | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|-----------|----|----|-----|-----|-----|
| Profile number | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 |
| Typical kbit/s | 9 | 17 | 25 | 44 | 41 | 49 | 66 | 84 | 101 | 119 | 136 |

The “HiFi Voice” Profiles

When extremely high quality speech is required, use the “HiFi Voice” profiles.

These profiles support Vox and Pause. However, Vox works best with the “Voice” and “Wideband Voice” profiles.

| HiFi Voice | | | | | | |
|-----------------------|----|----|----|----|----|-----------|
| Profile number | 00 | 01 | 02 | 03 | 04 | 05 |
| Typical kbit/s | 36 | 49 | 59 | 71 | 79 | 87 |

The “Music” Profiles

The “Music” profiles are intended for HiFi music, and are capable of offering very high-quality stereo sound.

These profiles don't support Vox and Pause.

| Music | | | | | | |
|-----------------------|----|----|----|----|-----|------------|
| Profile number | 00 | 01 | 02 | 03 | 04 | 05 |
| Typical kbit/s | 44 | 63 | 80 | 99 | 117 | 135 |

2.3 Running the VS1053b Ogg Vorbis Encoder

This chapter will describe how to load and run the Ogg Vorbis Encoder. A C language source file `vs1053oggrec.c` has been included in this package as an example.

2.3.1 VS1053b Ogg Vorbis Encoder Registers

| Register | Bits | Description |
|-------------|------|---|
| SCI_MODE | 14 | Select MIC/LINE1 |
| | 12 | Set to 1 when told in the instructions |
| | 1 | VU meter stereo mode activation |
| SCI_AICTRL0 | 15:0 | Maximum signal level, set to 0x8000 (mono) or 0x8080 (stereo) |
| SCI_AICTRL1 | 15:0 | Recording gain (1024 = 1×) or 0 for automatic gain control |
| SCI_AICTRL2 | 15:0 | Maximum autogain amplification (1024 = 1×, 65535 = 64×) |
| SCI_AICTRL3 | 0 | W: Finish recording, set to 0 |
| | 1 | R: Recording finished, set to 0 |
| | 2 | R: There is at least one byte to read, set to 0 |
| | 3 | W: Input channel select (only mono profiles), 0 = left, 1 = right |
| | 7:4 | W: Max samples in frame = $n \times 4096$, 0 = no limit |
| | 15:8 | R: The next data byte if available, set to 0 |

Before activating Ogg Vorbis recording, the user **must** initialize registers SCI_AICTRL0 - SCI_AICTRL3. SCI_AICTRL1 and SCI_AICTRL2 can be altered during recording.

SCI_AICTRL0 records the maximum absolute value of the signal. The maximum value of this linear register is 0x7FFF in mono mode, or 0x7F7F in stereo mode. For more information on how to use this register, see Chapter 2.4.1. SCI_AICTRL0 is updated once for every Vorbis block, i.e. 10...170 times per second with the current profiles.

SCI_AICTRL1 controls linear recording gain. 1024 is equal to digital gain 1 (recommended for best quality), 2048 is equal to digital gain 2, and so on. If the user wants to use automatic gain control (AGC), SCI_AICTRL1 should be set to 0. Speech applications are often better off using some AGC, as this helps to get relatively uniform speech loudness in recordings.

SCI_AICTRL2 controls the maximum AGC gain if SCI_AICTRL1 is set to 0. This limits amplification of noise when there is no signal. For more information on recording levels, see Chapter 2.4.

SCI_AICTRL3 offers run-time controls, channel selection for mono input, the maximum number of samples that are allowed in one Ogg frame with 4096 sample granularity, and the next unread byte, if any. Normally 0 (no limit) is a good value for granularity, but streaming applications often have maximum delay considerations, thus making limiting necessary. Note that limiting frame size isn't entirely accurate, so always select a maximum frame size slightly lower than your absolute upper delay limit.

Example: your samplerate is 44100 Hz and you want frame output at least every 500 ms. Then you'll need samples at 22100 sample intervals. Round this downwards to the closest 4096 multiple, and you'll get 16384 samples, so you'll write 0x40 to SCI_AICTRL3. The bitrate penalty of this example is ≈ 0.5 kbit/s.

For this to work your reads must be byte aligned as explained in Chapter 2.3.3.

For Vox operation controls, see Chapter 2.5.

2.3.2 Loading and Starting the Code

To load and start the VS1053b Ogg Vorbis Encoder, do the following steps:

1. Set the VS1053b clock to the highest value just below 55.3 MHz. Example: if the external clock is 12.288 MHz, this can be done by setting the clock to 4.5X, or 55.296 MHz by writing 0xC000 to register SCI_CLOCKF (3). If you use another input clock, adjust SCI_CLOCKF accordingly.
Examples: Value for 12 MHz is 0xC3e8, 12.288 MHz is 0xC000, 13 MHz is 0xA4E2.
2. Set SCI_BASS (2) to 0.
3. Reset VS1053b by setting bit SM_RESET (2) to 1 in register SCI_MODE (0). At the same time, clear bit SM_ADPCM (12) to 0. Remember that resetting takes time, so don't do any further register writes before the DREQ pin has risen!
4. Disable all interrupts except the SCI interrupt by writing 0x2 to VS1053's internal register VS1053_INT_ENABLE. This is done by first writing 0xC01A to SCI_WRAMADDR (7), then 0x2 to SCI_WRAM (6).
5. Load the plugin profile you intend to run (Chapter 4). Note that you will have to redo this each time you want to activate recording.
6. Set bit SM_ADPCM (12) in register SCI_MODE (0) to 1. At the same time, you can also select LINE input instead of MIC input by setting bit SM_LINE1 (14) to 1. If you want to use the VU meter in stereo mode, set also bit SM_LAYER12 (1) to 1. Do *not* set SM_RESET (2) at the same time!
7. Set recording level control registers SCI_AICTRL1 (13) and SCI_AICTRL2 (14). Typical good values for conservative AGC are 0 and 4096, respectively. For a HiFi application, recommended values are 1024 and 0, respectively.
8. If you want to use a VU meter, write 0x8000 to SCI_AICTRL0 (12) if using a mono profile or 0x8080 is using a stereo profile.
9. Set a proper value (often 0) to SCI_AICTRL3 (15).
10. If you use a profile that support Vox, set Vox registers. For details, see Chapter 2.5.
11. Activate the encoder by writing 0x34 to register SCI_AIADDR (10).
12. Wait until DREQ pin is high before reading any data.

2.3.3 Reading Ogg Vorbis Data

After Ogg Vorbis recording has been activated, registers SCI_HDAT0 and SCI_HDAT1 have new functions.

The Ogg Vorbis bitstream buffer size is 4096 16-bit words, or 8 KiB. The fill status of the buffer can be read from SCI_HDAT1. If SCI_HDAT1 is greater than 0, you can read that many 16-bit words from SCI_HDAT0.

If data is not read fast enough from SCI_HDAT0, the buffer overflows and returns to empty state. A data overflow will result in an incorrect file that may be undecodable. However, because of the large size of the bitstream buffer and the relatively slow bitrates of Ogg Vorbis, this situation should be easy to avoid.

Ogg frames are byte aligned. Because of the 16-bit SCI interface this may be a problem for streaming applications where it is important to forward any Ogg frame as soon as possible: the latest frame may pend upon its last byte. If transferring data with a low delay is important, you can read whether there is one extra data byte by reading SCI_AICTRL3 *twice in a row*. The result of the latter read will contain the correct data: if bit 2 is 1, then bits 15:8 contain the odd byte. After reading the odd byte successfully, discard bits 15:8 of the next read from SCI_HDAT0 because they contain the same data. (NOTE: if you don't stream or don't need the shortest possible delay, you can ignore this paragraph!)

If you are having trouble with receiving data, notice that all Ogg Vorbis files always begin with the following 4 bytes: 0x4f 0x67 0x67 0x53 (the string "OggS"). If you get 0x67 0x4f 0x53 0x67 ("gOSg") instead, you are storing the least and most significant byte in the 16-bit data words incorrectly. If you get 0x00 0x00 0x67 0x53, you have read data from SCI_HDAT0 too soon after starting the application.

2.3.4 Reading Additional Data while Recording Ogg Vorbis

You can get extra side information while recording Ogg Vorbis data to see whether VS1053b is still working. The following VS1053b X memory addresses may be read for extra data:

| X Memory Address | Description |
|------------------|--------------------------------------|
| 0x8 | 16 LSb's of recording time (seconds) |
| 0x9 | 16 MSb's of recording time (seconds) |
| 0xC | 16 LSb's of average bitrate (bits/s) |
| 0xD | 16 MSb's of average bitrate (bits/s) |
| 0x1800 | 16 LSb's of sample counter |
| 0x1801 | 16 MSb's of sample counter |

Example: To read the average bitrate, do as follows. First write 0xC to SCI_WRAMADDR (7). Then read from SCI_WRAM (6). This is 16 least significant bits of the bitrate. Then read the 16 more significant bits by reading SCI_WRAM again.

You can read the recording time as a sanity check that VS1053b is working: if the register contents don't change every second, you'll have to take protective measures.

2.3.5 Finishing Ogg Vorbis Recording

To create fully compliant Ogg Vorbis files, Ogg Vorbis bitstreams need to be shut down properly. The following algorithm can be used to implement this:

1. Set bit 0 of SCI_AICTRL3 (15) to 1.
2. Continue reading data through SCI_HDAT0 and SCI_HDAT1 as usual, but check SCI_AICTRL3's bit 1 from time to time. When this bit turns to 1, the Ogg Vorbis encoder has finished writing to the buffer.
3. Write the remaining words from the bitstream buffer as normal using SCI_HDAT0 and SCI_HDAT1.
4. Read SCI_AICTRL3 *twice* (changed from version 1.4!) and check whether bit 2 is 1 in the latter read. If it is, then don't write to the file the last byte (bits 7:0) of the last word that have come through SCI_HDAT0.
5. Reset VS1053b to normal state using software reset. Remember to clear register SCI_MODE (0) bit SM_ADPCM (12) if you don't wish to start ADPCM recording. Remember to also set bit SM_LAYER12 (1) to an appropriate value.
6. If you want to restart recording, you have to completely reload and restart the recording application.

2.4 Recording Levels and Automatic Gain Control (AGC)

The VS1053b Ogg Vorbis Encoder offers signal level monitoring through SCI_AICTRL0. It is recommended that devices that offer recording would show a signal level in a decibel scale. For the decibel scale, see Chapter 2.4.4. This can be done by showing and clearing SCI_AICTRL0 contents at regular intervals.

A good VU meter should be implemented in such a way that it visually advises the user to avoid using the last 6 dB of the available dynamic range.

2.4.1 Reading the Recording Level

The recording level meter has two settings: mono and stereo. You can use mono mode with both mono and stereo streams, but you should use the stereo mode only with stereo profiles. The new meter is compatible with versions pre 1.60 if SCI_MODE bit SM_LAYER12 is cleared first.

Recording Level Meter: Mono Mode

Activate mono mode by clearing SCI_MODE (0) register bit SM_LAYER12 (1) to 0.

To read the level, repeat the following loop:

- Write 0x8000 to SCI_AICTRL0.
- Wait for at least 1/50 s. Note: Constant reading of side information causes load on the VS1053 and may cause unexpected crackles in sound.
- Check whether SCI_AICTRL0 & 0x8000 is 0. If not, wait a little more and read again.
- Use the value as explained in this Chapter. Repeat the loop.

Recording Level Meter: Stereo Mode

Activate stereo mode by setting SCI_MODE (0) register bit SM_LAYER12 (1) to 1.

To read the left and right channel levels, repeat the following loop:

- Write 0x8080 to SCI_AICTRL0.
- Wait for at least 1/50 s. Note: Constant reading of side information causes load on the VS1053 and may cause unexpected crackles in sound.
- Check whether SCI_AICTRL0 & 0x8080 is 0. If not, wait a little more and read again.
- Left channel value is SCI_AICTRL0 & 0x7F00.
- Right channel value is (SCI_AICTRL0 & 0x7F) × 256.
- Use the values as explained in this Chapter. Repeat the loop.

2.4.2 Setting AGC

When the highest dynamic range and sound fidelity is required, AGC should be turned off and recording gain should be set to 1 (SCI_AICTRL1 = 1024). A good example of this would be music recording. This is particularly recommended for the “HiFi Voice” and “Music” profiles, although there might be cases where recording level control would be needed even with these cases.

However, in some cases it is required that the audio dynamic range is compressed. An example of such a case is when a device should retain a uniform recording level of a discussion of several people or of one person moving closer and further from the recording device. In such a case, it is a good idea to use AGC.

The AGC unit adjusts signal power in such a way that the maximum sample value from a sine wave would become as close to 16300 as possible. If the signal is too strong, recording level is decreased, and vice versa. The maximum recording level can be set with register SCI_AICTRL2.

When AGC is used, conservative maximum gain values often give the best sounding results. Example: 12 dB (SCI_AICTRL1 = 0, SCI_AICTRL2 = 4096). In some cases more extreme values may help to make quiet speech more intelligible, but such values may also add excessive background noise and make sound quality less pleasing.

2.4.3 Building a Useful VU Meter

In an encoder application, if the recording level is too low, extraneous background noise may be introduced to the sound. Conversely, if the recording level is so high that the highest values cannot be represented numerically, signal clipping occurs, and this may cause severe distortion to sound.

In a recording device, it is useful to have a VU meter that shows the signal level so that both too low signal levels and clipping is avoided. This is very important so that the user has a chance to either adjust the recording or input signal level.

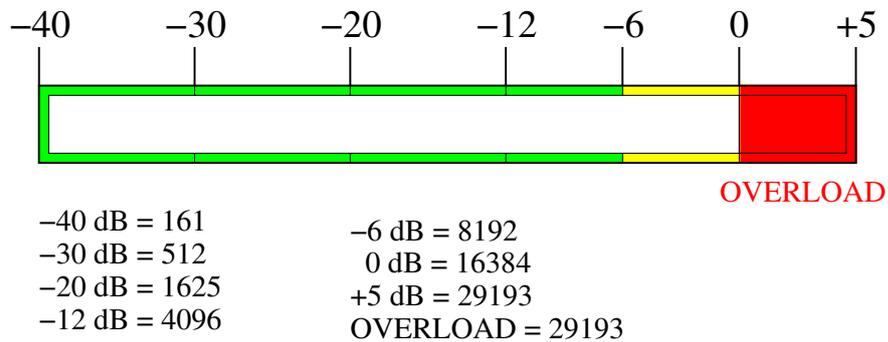


Figure 1: Example colour VU meter.

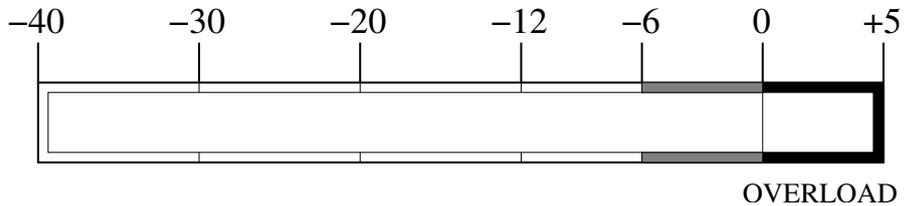


Figure 2: Example monochrome VU meter.

Figures 1 and 2 show example VU meters. The 0 dB reference point has been set to signal level 16384, which is one half of the maximum amplitude, leaving a 6 dB headroom for the device.

The lowest signal level shown is a choice that can vary a lot depending on the application. While -40 dB is a high-fidelity favourite, a VU meter will work just as well with a lower limit of -20 dB. If display space is scarce, low limit could even be set to -12 dB.

Between -6 and +6 dB the VU meter precision should preferably be 1 dB, and at most 2 dB. Below -12 dB step size can be several decibels.

The important thing in a VU meter is to visually show the user that it is not recommended to regularly use the highest 6 dB of the recording scale, and that an overload situation ($\geq +5$ dB, or over linear value 29193) is an error condition. To help this the OVERLOAD symbol should be kept lit or blinking for at least 2 seconds each time an overload situation occurs. Many of these cues are often ignored in digital VU meters, and partly as a result of this even professional recordings are all too often made at recording levels that destroy signal integrity.



Figure 3: Real record display with monochrome VU meter.

Figure 3 shows a real implementation of a recording display that uses a Monochrome VU Meter. Information shown on the top line includes recording time, free disc space, overload warning and a bitrate counter. All this data except from the free disc space number have been obtained from the VS1053b Ogg Vorbis Encoder.

Recording gain and AGC can be set at the center of the screen if a profile is used that support these functions. However, they are greyed out in this picture because they are not available in the Stereo Music profile that has been used.

The VU Meter is at the bottom. The solid line is the current recording level as read from SCI_AICTRL0 and converted to Decibel scale as shown in Chapter 2.4.4. The greyed line is the top level of the last 3 seconds. If the greyed line $\geq +5$ dB, the orange OVERLOAD message blinks twice a second for three seconds.

2.4.4 Converting from Linear to Decibel Scale

To convert from linear to dB scale, use the formula

$$dB = 20 \times \log_{10}\left(\frac{m}{32768}\right) + 96$$

where dB is the result and m is a value returned by the Recording Level Meter.

To implement a good approximation of this formula from linear to dB scale on architectures where multiplications and logarithms are expensive operations, the following code can be used:

```

const unsigned short linToDBTab[5] = {36781, 41285, 46341, 52016, 58386};

/*
  Converts a linear 16-bit value between 0..65535 to decibels.
  Reference level: 32768 = 96dB (largest VS1053b number is 32767 = 95dB).
  Bugs:
  - For the input of 0, 0 dB is returned, because minus infinity cannot
    be represented with integers.
  - Assumes a ratio of 2 is 6 dB, when it actually is approx. 6.02 dB.
*/
unsigned short LinToDB(unsigned short n) {
  int res = 96, i;

  if (!n) /* No signal should return minus infinity */
    return 0;

  while (n < 32768U) { /* Amplify weak signals */
    res -= 6;
    n <<= 1;
  }

  for (i=0; i<5; i++) /* Find exact scale */
    if (n >= linToDBTab[i])
      res++;

  return res;
}

```

2.4.5 vs1053oggrec.c

The file `vs1053oggrec.c`, quoted below, has also been included in the VS1053b Ogg Vorbis Encoder package.

```
/*
```

```
Example code how to implement VS1053 Ogg Vorbis recording using  
the VS1053 Ogg Vorbis Recording plugin. The code is generic enough  
so that it should pretty much compile and run on any microcontroller.
```

```
The code makes the following assumptions:
```

- `<stdio.h>` functions `fopen()`, `fclose()`, `fgetc()` and `fputc()` are available
- The following VS1053 interfacing functions must exist:
 - `void Write1053Sci(int regNo, u_int16 value)`: writes value to SCI register
 - `u_int16 Read1053Sci(int regNo)`: reads value from SCI register
 - `WaitFor1053Dreq(int timeOut)`: Waits for 1 microsecond, then until DREQ goes up or timeOut is reached (timeOut=1 is 10 ms).
- `EndRecording()` function must exist, which returns non-zero when the user has requested that recording should be finished.

```
NOTE!
```

```
This code serves as an example of how to create generic C microcontroller  
code for VS1053 recording.
```

```
Author: Henrik Herranen / VLSI Solution 2011.
```

```
Copyright: Use freely for any project with any VLSI Solution's IC.
```

```
Warranty: Absolutely none whatsoever.
```

```
*/
```

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>
```

```
/* Change these if they don't match you architecture. */
```

```
typedef unsigned short u_int16;  
typedef short          s_int16;  
typedef unsigned long  u_int32;  
typedef long           s_int32;
```

```
#define SCIR_MODE          0  
#define SCIR_STATUS       1  
#define SCIR_BASS         2  
#define SCIR_CLOCKF       3  
#define SCIR_DECODE_TIME  4  
#define SCIR_AUDATA       5  
#define SCIR_WRAM         6  
#define SCIR_WRAMADDR     7  
#define SCIR_HDAT0       8  
#define SCIR_HDAT1       9  
#define SCIR_AIADDR      10  
#define SCIR_VOL         11  
#define SCIR_AICTRL0     12
```

```
#define SCIR_AICTRL1    13
#define SCIR_AICTRL2    14
#define SCIR_AICTRL3    15

/* returns minimum of two given numbers */
#define min(a,b) (((a)<(b))?a):(b))

/* Loads an image file into VS1053 from a file. */
auto u_int16 SpiLoadImageInto1053(FILE *fp);

/* Main recording program */
void main(void) {
    FILE *outFP, *inFP;
    u_int16 pluginStartAddr;
    u_int16 state = 0;
    u_int16 i;
    u_int16 wordsToRead;
    u_int16 wordsWaiting;

    /* Perform Microcontroller specific initializations here. */

    /* Following VS1053 operations are according to the
       VS1053b Ogg Vorbis Encoder application manual. For
       details read Chapter Loading and Starting the Code. */

    /* Set VS1053 clock to 4.5x = 55.3 MHz */
    Write1053Sci(SCIR_CLOCKF, 0xC000);
    WaitFor1053Dreq(TIMER_TICKS/10);

    /* Clear SCI_BASS */
    Write1053Sci(SCIR_BASS, 0);

    /* Reset VS1053 */
    Write1053Sci(SCIR_MODE, SMF_SDINew | SMF_RESET);
    WaitFor1053Dreq(TIMER_TICKS/10);    /* Wait until DREQ is high or 100 ms */

    // Disable all interrupts except SCI
    Write1053Sci(SCIR_WRAMADDR, VS1053_INT_ENABLE);
    Write1053Sci(SCIR_WRAM, 0x2);

    /* Load the recorder application to VS1053
       This source code uses .img image files for loading.
       If you use .plg files, use the source as described
       in the VS1053b Ogg Vorbis Encoder application manual. */
    inFP = fopen("venc44k2q05.img", "rb");
    if (!inFP)
        goto end;
    pluginStartAddr = SpiLoadImageInto1053(inFP);
    fclose(inFP);
}
```

```
/* If loading failed, give up. */
if (pluginStartAddr == 0xFFFF)
    goto end;

/* Now open output file. It's better to do this before activating
   recording so that even if opening the output file is slow,
   you will not lose audio data. */
outFP = fopen("record.ogg", "wb");
if (!outFP)
    goto end;

/* Set VS1053 mode bits as instructed in the VS1053b Ogg Vorbis Encoder
   manual. Note: for microphone input, leave SMF_LINE1 unset! */
Write1053Sci(SCIR_MODE, SMF_LINE1 | SMF_ADPCM | SMF_SDINew);

/* Rec level: 1024 = 1. If 0, use AGC */
Write1053Sci(SCIR_AICtrl1, 1024);
/* Maximum AGC level: 1024 = 1. Only used if SCI_AICtrl1 is set to 0. */
Write1053Sci(SCIR_AICtrl2, 0);
/* Miscellaneous bits that also must be set before recording. */
Write1053Sci(SCIR_AICtrl3, 0);

/* Activate recording from the address we got. (In the case of the Ogg
   Vorbis Encoder application, pluginStartAddr = 0x34.) */
Write1053Sci(SCIR_AIADDR, pluginStartAddr);
WaitFor1053Dreq(TIMER_TICKS/10);

/*
   By now we have:
   - Initialized the VS1053
   - Loaded the VS1053 recording plugin
   - Opened the output file
   - Activated the VS1053 recording plugin

   So, now it's time to record. Below is the recording loop.

   The variable "state" controls the progression of recording as follows:
   state = 0: Normal recording.
   state = 1: User has requested end of recording, microcontroller has
       also requested this from the VS1053 recording plugin.
   state = 2: The VS1053 plugin has stopped recording, but the
       microcontroller is still collecting data from the VS1053 buffers.
   state = 3: Recording finished.
*/

/* Main loop */
while (state < 3) {

    /* Check when to end recording. The function EndRecording() is
       microcontroller specific and should be implementing separately
       for each application. */
```

```
if (EndRecording() && !state) {
    state = 1;
    Write1053Sci(SCIR_AICTRL3, 1); // Send VS1053 request to stop recording
}

/* See how many 16-bit words there are waiting in the VS1053 buffer */
wordsWaiting = Read1053Sci(SCIR_HDAT1);

/* If user has requested stopping recording, and VS1053 has
stopped recording, proceed to the next state. */
if (state == 1 && Read1053Sci(SCIR_AICTRL3) & (1<<1)) {
    state = 2;
    /* It is important to reread the HDAT1 register once after
VS1053 has stopped. Otherwise there is the chance that
a few more words have just arrived although we just
read this register. So, do NOT optimize the following
line away! */
    wordsWaiting = Read1053Sci(SCIR_HDAT1);
}

/* Read and transfer whole 512-byte (256-word) disc blocks at
the time. The only exception is when recording ends: then
allow for a non-full block. */
while (wordsWaiting >= ((state < 2) ? 256 : 1)) {
    wordsToRead = min(wordsWaiting, 256);
    wordsWaiting -= wordsToRead;

    /* If this is the very last block, read one 16-bit word less,
because it will be handled later separately. */
    if (state == 2 && !wordsWaiting)
        wordsToRead--;

    /* Transfer one full data block, or if this is the very last
block, all data that's left except for the last word. */
    {
        u_int16 t;
        u_int16 i;
        for (i=0; i<wordsToRead; i++) {
            t = Read1053Sci(SCIR_HDAT0);
            fputc(t >> 8 , outFP);
            fputc(t & 0xFF, outFP);
        }
    }

    /* If this is the last data block... */
    if (wordsToRead < 256) {
        u_int16 lastWord;
        state = 3;

        /* ... read the very last word of the file */
        lastWord = Read1053Sci(SCIR_HDAT0);

        /* Always write first half of the last word. */
        fputc(lastWord >> 8 , outFP);
    }
}
```

```
        /* Read twice SCIR_AICTRL3, then check bit 2 of latter read. */
        Read1053Sci(SCIR_AICTRL3);
        if (!(Read1053Sci(SCIR_AICTRL3) & (1<<2))) {
            /* Write last half of the last word only if bit 2 is clear. */
            fputc(lastWord & 0xFF, outFP);
        }
    } /* if (wordsToRead < 256) */
} /* while (wordsWaiting >= ((state < 2) ? 256 : 1)) */

} /* while (state < 3), end of main loop */

/* That's it! We've perfectly recorded an Ogg Vorbis file, so now
   we only need to close the file and be happy about it. */
fclose(outFP);

end:
/* Finally, reset VS1053 so that we will hear no more monitor audio */
Write1053Sci(SCIR_MODE, SMF_SDINER | SMF_RESET);

/* End up in infinite loop */
while (1)
    ;
}
```

2.5 Vox and Pause Operation

Vox / Pause allows the user to temporarily pause recording either based on audio content or direct user control. (For Ogg Vorbis Encoder Profiles that support Vox / Pause operation, see Chapter 2.2.1.)

Vox control allows the unit to record only when someone is actually speaking. An audio buffer makes it so that recording resumes slightly “before” the moment when speech is detected, so in typical cases no syllables are lost at the beginning of a sentence.

Pause override allows the user to set and clear pause mode at will. This allows the user to create a system where pause mode may be activated with a pause switch, like in a traditional tape recorder.

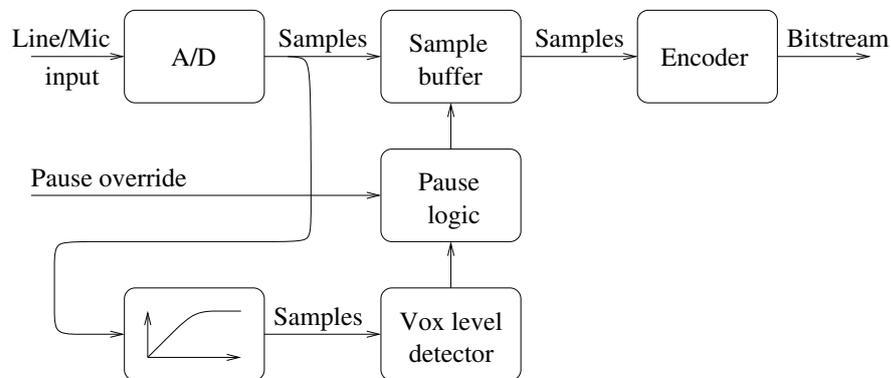


Figure 4: Vox / Pause logic.

Figure 4 presents the pause logic. Input samples are fed to the Vox detector through a high-pass filter. If pause hasn’t been overridden by the user, the Vox detector level is used to control pause mode.

Note: It is not recommended to use Vox with a streaming system. When pause is activated either by the user or by Vox, the sample counter (Chapter 5.1) stops and no Ogg frames, not even empty ones, are output by the Encoder.

2.5.1 Vox and Pause Registers

| X Mem Addr | Name | Default | Description |
|------------|------------|---------|--|
| 0x1808 | OffLimit | 0x18 | Signal limit for Vox pause |
| 0x1809 | OnLimit | 0x20 | Signal limit for Vox record |
| 0x180A | OffTimeLSW | 0x7700 | 16 LSB's of pause delay time in 1/48000 s |
| 0x180B | OffTimeMSW | 0x1 | 16 MSb's of pause delay time in 1/48000 s |
| 0x180F | PauseCtl | 0x2 | Vox control / Pause override 0 = Automatic Vox operation 1 = forbidden 2 = Pause override: Normal recording 3 = Pause override: Pause on |

If the signal after the high-pass filter is below OffLimit for more than OffTime 48 kHz samples, and if PauseCtl = 0, or if PauseCtl = 3, then pause mode is activated. Default value of OffTime = 0x17700 = 96000 equals two seconds. Note that OffTime is always calculated in the 48 kHz domain regardless of the samplerate of the Ogg Vorbis Encoder Profile.

If even a single sample goes above OnLimit, and if PauseCtl = 0, or if PauseCtl = 2, then pause mode is deactivated and normal recording continues.

OffLimit and OnLimit are linear values. Thus, doubling the value lifts the limit by 6 dB. Having two separate values for the limits allow for hysteresis between stopping and restarting recording. OnLimit must always be greater than or equal to OffLimit.

Example: To set OffTime, do as follows. First write 0x180A to SCI_WRAMADDR (7). Then write 16 least significant bits of OffTime (e.g. 0x7700) to SCI_WRAM (6). Then write the 16 most significant bits by writing (e.g. 0x1) to SCI_WRAM (6) again.

Vox and Pause registers may be written to while the Ogg Vorbis Encoder is active.

2.5.2 Vox Output to GPIO4

It is optionally possible for the user to optionally see the current status of Vox in GPIO4. To activate this option, do the following steps before activating the Ogg Vorbis Encoder:

- Write 0xC017 to SCI_WRAMADDR (7).
- Read value x from SCI_WRAM (6).
- Set $x = x|16$;
- Write 0xC017 to SCI_WRAMADDR (7).
- Write new value x to SCI_WRAM (6).

After this operation, whenever GPIO4 is low, normal recording is done. When GPIO4 is high, then pause mode (either by Vox or user control) is active. So, whenever GPIO4 goes high, the main controller may read all data that is left in the buffer, and set itself to sleep for as long until GPIO4 goes low again.

2.6 Samplerate Considerations

The Ogg Vorbis encoder cannot always work at exactly the right samplerate. To be exact, the samplerate is accurate only when the input clock is 12.288 MHz and the nominal samplerate of a profile is either 8000 Hz or 16000 Hz. In all other cases the samplerate can be calculated as follows. First let:

- f_i = input clock divided by 4000 (same value that you write to SC_FREQ (bits 10:0) of register SCI_CLOCKF, or 3072 if you have written 0 to these bits to indicate 12.288 MHz clock).
- f_n = nominal samplerate of the profile (8000, 16000 or 44100).
- $r = 3$ if $f_n = 8000$, otherwise $r = 2$.
- Now let $u = f_n \times r$.
- Let $d = \lfloor \frac{f_i \times 2000 + \frac{u}{2}}{u} \rfloor$
- Now our real samplerate $f_s = \frac{2000 \times f_i}{d \times r}$

Example: We are running at an input clock of 12.288 MHz and we are running the Stereo Music profile that has a nominal samplerate of 44100 Hz.

First $f_i = \frac{12288000}{4000} = 3072$.

Because $f_n = 44100$, then $r = 2$.

Thus $u = 44100 \times 2 = 88200$.

This leads to $d = \lfloor \frac{3072 \times 2000 + \frac{88200}{2}}{88200} \rfloor = 70$

And, finally, our real samplerate $f_s = \frac{2000 \times 3072}{70 \times 2} \approx 43885.7$ (Hz)

As can be seen, in this example the nominal and real samplerate have a difference of 0.5%. While this is normally not significant, it is good to know and important in the case of streaming.

2.7 Post-Processing the Recording with VorbisGain

It is recommended to post-process your own recording later on a PC machine using VorbisGain to get equal loudness with other recordings. For details, see Chapter 3.3.

3 The Ogg Vorbis Format

This chapter gives a brief introduction to the Ogg Vorbis format and presents recommendations for basic parameters when transcoding MP3 files or compressing CD content into Ogg Vorbis files on a PC. The intent is to give VLSI Solution's customers information on how to create high-quality Ogg Vorbis content.

Before using the information in this document, you should get the free Ogg Vorbis encoder "oggenc", minimum version 1.0.2 (2005), or a version with aoTuV tunings. While the instructions will work with older versions, sound quality will not be as good. Download oggenc at <http://www.vorbis.com/>.

It is also strongly encouraged to get VorbisGain, a program that tags files with loudness information (Chapter 3.3). For this example, vorbisgain v0.36 has been used. Download VorbisGain at <http://www.sjeng.org/vorbisgain.html>.

3.1 Introduction to Ogg Vorbis

Ogg Vorbis is a license-free audio codec that allows for high-quality sound around a large gamut of bit-rates, ranging from roughly 10 to 500 kbit/s.

With a given bit-rate, Vorbis is capable of surpassing the quality of MP2, MP3, WMA and AAC LC, particularly in low-bit-rate applications.

An Ogg Vorbis file is a Vorbis compressed file that has been encapsulated inside an Ogg stream. The default file extension for Ogg Vorbis files is ".ogg".

An Ogg Vorbis file consists of two major parts: header and actual data. Every Ogg Vorbis file begins with a header which contains compression tables that the decoder needs to correctly decode first. After the header has been fully received, Vorbis audio may be decoded. The size of the header information is around 4 KiB for most encoders. (However, for VLSI Solution's VS1053b encoder, header size is only roughly 1.5 KiB.)

After the header has been decoded, full random access is available in an Ogg Vorbis file. As opposed to many other formats, an Ogg Vorbis file contains its own time code, so absolute position in a file can be displayed after random access operations (fast forward / rewind).

Ogg Vorbis is not particularly well suited for encoding very short, low bit-rate samples. E.g. a 16 kbit/s audio sample that lasts for one second would take 2 KiB for the actual audio data, so the default 4 KiB header triples the file size. However, for normal applications where audio lasts for more than 10 seconds, this header overhead is negligible.

3.1.1 Variable Bit-Rate

For a long time almost all MP3 files had a constant bit-rate, typically 128 kbit/s. While constant bit-rate (CBR) makes it easy to calculate a relation between file size/position

and song length, and while it makes it easier to stream files, CBR is not optimal for quality. When there is a difficult passage in music, there may not be enough bits available to encode audio flawlessly. Conversely, in passages that are easier to encode there may be bits left unused because the encoder didn't need all the bits available in the bitstream.

Where CBR tries to keep the bit-rate constant regardless of content, Variable Bit-Rate (VBR) tries to keep quality constant. This will have the effect of changing the bit-rate of the file depending on how difficult a particular portion of music is to encode. Because sound quality is always kept constant, VBR has a bit advantage to CBR. Thus, VBR allows either for smaller files with the same perceived sound quality or equally sized files with better perceived sound quality.

Ogg Vorbis is by nature a variable bit-rate format and works best when this feature is not limited. Depending on the material the bit-rate can have significant fluctuations: e.g. songs that have been compressed with a quality setting with a nominal bit-rate of 96 kbit/s can easily have average bit-rates between 80 to 105 kbit/s, or even more with extreme material.

3.1.2 Transcoding

Transcoding is the process of converting from one lossy format to another, like from MP3 to Ogg Vorbis.

It is important to understand that although Ogg Vorbis is a better format than MP3, transcoding never enhances sound quality. What has been lost in the initial MP3 compression stage, will stay lost when converting to Ogg Vorbis, and there will always be some additional quality loss. While the extra quality loss can be small enough that it may be impossible to tell, the end result still is never better than the original file was.

If possible, it is recommended that users should avoid transcoding to avoid generation loss. If transcoding is done (e.g. to get smaller files for a portable device), it is a good idea to keep the original files.

For best results, compress Ogg Vorbis files from uncompressed sources, like CDs.

3.2 Compressing into Ogg Vorbis with a PC

The whole process of getting Ogg Vorbis files looks roughly like this:

- Preparation: Decode an MP3 / WMA / AAC etc file or preferably rip a CD to WAV format and make sure that you have “oggenc” (Chapter 3.2.1).
- Compression to Ogg Vorbis (Chapters 3.2.2 and 3.2.3).
- Optional post-processing with VorbisGain (Chapter 3.3).

3.2.1 Preparation

If you intend to transcode (=recompress) MP3 files, you first have to convert them to 16-bit RIFF WAV format (usually known for a “.wav” file name extension). However, before you do that, please have a look at Chapter 3.1.2 for details on transcoding.

If you encode CDs, you have to make sure they are read initially into WAV format. It is recommended to use CD sources for compression if possible.

When you have audio in WAV format, you can use *oggenc* to compress files, either in stereo or mono.

Before continuing to the next stage, make sure you have the free Ogg Vorbis encoder “oggenc”, minimum version 1.0.2 (2005).

3.2.2 Compressing CD-Quality Music

If we assume the input file is called file.wav and the output is to be called file.ogg, the following parameters may be used.

It is assumed that the input audio is sampled at the CD 44.1 kHz sample rate. While these commands will also work for other samplerates, quality and file sizes may differ.

For each quality setting, the table shows the command needed to convert the file, a typical bit-rate for that quality, how many hours of music could be stored to a 1 gibibyte memory card, and a short description of the mode. For comparison purposes, also an uncompressed CD bitstream is shown in the table.

| CD music compression | | | |
|-----------------------|-------------|--------------|--|
| Command | Typ. kbit/s | Hours/ 1 GiB | Description |
| oggenc -q -1 file.wav | 45 | 53 | Smallest files size, better than 96 kbit/s MP3 |
| oggenc -q 0 file.wav | 64 | 37 | Ok quality, equals 112 kbit/s MP3 |
| oggenc -q 2 file.wav | 96 | 25 | Better than most 128 kbit/s MP3 files |
| oggenc -q 4 file.wav | 128 | 19 | Often CD transparent |
| oggenc -q 6 file.wav | 192 | 12 | CD quality almost always |
| CD | 1411 | 1.7 | Uncompressed CD |

Graphical front-ends usually contain one or several CBR bit-rate slider(s) and a quality slider that usually goes from -1 to 10. CBR sliders should be deactivated, and the quality slider should be set to the “-q” value presented in the table.

3.2.3 Compressing Speech

Speech requires different coding parameters than music. First, speech can usually be coded in mono, so all the following examples convert audio to mono. Also, to get smaller files, speech bandwidth can usually be suppressed without adversely affecting intelligibility.

The following table shows the command needed to convert a speech file called file.wav, a typical bit-rate for that quality, how many hours of music could be stored to a 1 gibibyte memory card, and a short description of the mode.

| Speech compression | | | |
|--|-------------|--------------|-----------------------------------|
| Command | Typ. kbit/s | Hours/ 1 GiB | Description |
| oggenc -downmix -resample 4000 -q -1 file.wav | 6 | 400 | Very low bandwidth |
| oggenc -downmix -resample 6000 -q -1 file.wav | 8 | 300 | Near mobile phone quality |
| oggenc -downmix -resample 8000 -q -1 file.wav | 9 | 265 | Slightly better than mobile phone |
| oggenc -downmix -resample 8000 -q 0 file.wav | 13 | 184 | Clean 3 kHz bandwidth sound |
| oggenc -downmix -resample 11025 -q -1 file.wav | 14 | 170 | Toy application quality |
| oggenc -downmix -resample 11025 -q 0 file.wav | 17 | 140 | Cleaner toy application |
| oggenc -downmix -resample 11025 -q 1 file.wav | 20 | 119 | Clean sound, 5 kHz bandwidth |
| oggenc -downmix -resample 16000 -q 1 file.wav | 24 | 99 | Wideband (7 kHz) sound |
| oggenc -downmix -resample 22050 -q 1 file.wav | 30 | 80 | 10 kHz bandwidth |
| oggenc -downmix -resample 44100 -q 0 file.wav | 36 | 66 | Full bandwidth (15+ kHz) |
| oggenc -downmix -resample 44100 -q 1 file.wav | 42 | 57 | Cleaner sound |
| oggenc -downmix -resample 44100 -q 2 file.wav | 48 | 50 | Close to transparent |
| oggenc -downmix -resample 44100 -q 5 file.wav | 80 | 30 | CD quality (but mono) |

The “-downmix” option converts sound to mono and the “-q” control works as in Chapter 3.2.2. Unfortunately, many graphical front-ends are missing the “-resample” option, which is a very powerful parameter for speech compression. If your program misses this slider, you either have to fill the missing values in a command option window or compress your files from oggenc’s command line mode.

3.3 Post-Processing with VorbisGain

Although the original CD standard defines a reference loudness that should be used for CD recording, current CDs no longer follow that standard. Because of this unequal loudness of CDs, songs compressed from CDs may have wildly differing subjective loudness, a phenomenon that is known to most users of portable digital audio players.

VorbisGain is a program that uses a psycho-acoustical model to determine an actual, perceived loudness of an audio track or album, and tags this information to the headers of an Ogg Vorbis file. This makes it possible for a VorbisGain aware decoder to adjust its output volume accordingly, and thus equalize loudness differences of songs without a need to recompress them.

Using VorbisGain on all Vorbis songs will enhance the user's listening experience because there is no need to turn volume up and down between songs. Many Ogg Vorbis playback programs and plug-ins for PC's as well as all VLSI Solution's Ogg Vorbis decoder chips are compatible with VorbisGain.

4 How to Load a Plugin

4.1 How to Load a .PLG File

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

The example vector 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 vector is in vector `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-- > 0) {
                WriteVS10xxRegister(addr, val);
            }
        } else { /* Copy run, copy n samples */
            while (n-- > 0) {
                val = plugin[i++];
                WriteVS10xxRegister(addr, val);
            }
        }
        i++;
    }
}
```

4.2 How to Load an .IMG File

The VS1053 image format is a bootable / loadable binary format with a three-byte header, followed by one or more boot records. The default image file suffix is “.IMG”.

The 3-byte header is shown below:

| VS1053 Boot Header, begins the file | | |
|-------------------------------------|-------|--------------------------|
| Name | Bytes | Description |
| HEADER | 3 | “P&H” (0x50, 0x26, 0x48) |

A boot record is shown below:

| VS1053 Boot Record, from 1 to n, last is always TYPE = Execute | | |
|--|-------|---|
| Name | Bytes | Description |
| TYPE | 1 | 0 = Instruction memory 1 = X data memory 2 = Y data memory 3 = Execute |
| L1 L0 | 2 | LEN = 256×L1 + L0 |
| A1 A0 | 2 | ADDR = 256×A1 + A0 |
| D1 D0 | LEN | DATA = 256×D1 + D0, repeat LEN/2 times |

The last boot record is always with TYPE = Execute. When an execute record is found, ADDR is the start address of the application.

The VS1053 boot sequence will skip any unknown records.

4.2.1 Example Microcontroller .IMG File Decoder

```
/* This support function that writes one word through SCI is needed. */
void WriteVS10xxRegister(unsigned short addr, unsigned short value);

#define TYPE_I 0
#define TYPE_X 1
#define TYPE_Y 2
#define TYPE_E 3

/* Returns either 0xFFFF for error or image file start address.
   For compactness does NOT check if fgetc() fails! */
auto u_int16 SpiLoadImageInto1053(FILE *fp) {
    s_int16 type;
    if (fgetc(fp) != 'P' || fgetc(fp) != '&' || fgetc(fp) != 'H')
        return 0xFFFF;

    while ((type = fgetc(fp)) >= 0 && type < 4) {
        static const u_int16 offsets[3] = {0x8000U, 0x0U, 0x4000U};
        u_int16 len, addr;

        /* Get length and address of the record */
        len = fgetc(fp) << 8; len |= fgetc(fp);
        addr = fgetc(fp) << 8; addr |= fgetc(fp);

        /* If execute record: return with start address */
        if (type == TYPE_E)
            return addr;

        /* Map address to WRAMADDR register space */
        if (type != TYPE_Y || addr < 0xE000)
            addr += offsets[type];

        /* Set write address */
        Write1053Sci(SCIR_WRAMADDR, addr + offsets[type]);

        /* Convert len from bytes to words, then write data */
        len >>= 1;
        while (len-- > 0) {
            u_int16 data = (u_int16)fgetc(fp) << 8;
            data |= fgetc(fp);
            Write1053Sci(SCIR_WRAM, data);
        };
    } /* while (type >= 0 && type < 4) */

    return 0xFFFF; /* Pass-through indicates error condition */
}
```

5 Building a Streaming System

This chapter presents how to build a streaming system over the Internet, RF link or other media.

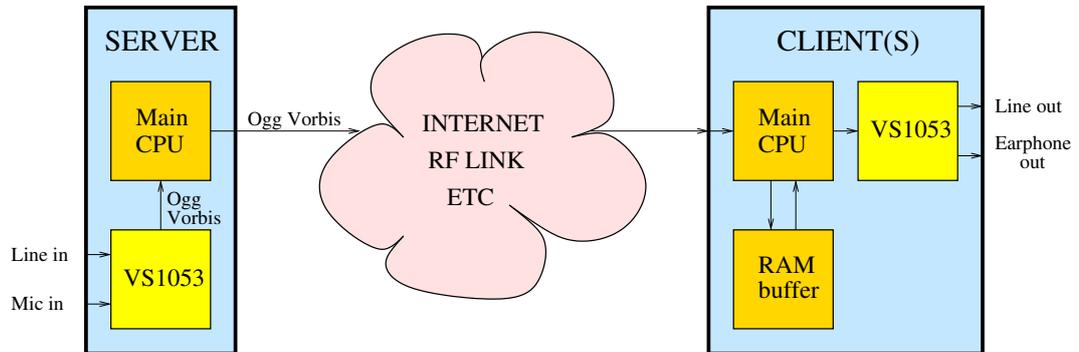


Figure 5: Streaming system using VS1053b.

Figure 5 presents an example system that uses a VS1053b for both encoding and decoding Ogg Vorbis audio.

First the audio data is fed to the VS1053b from a microphone or line input. It is encoded into Ogg Vorbis, which is sent by the Server's Main CPU over the transfer medium. To make sure Ogg Vorbis packets don't become too long, the Main CPU has set bits 7:4 in SCI_AICTRL3 to a sensible value prior to beginning encoding (Chapter 2.3.1).

The Main CPU on the Client end receives data and stores it into a RAM ring buffer (from e.g. 8 to 512 KiB depending on the application). It also transmits data to the VS1053b from the other end of the ring buffer. If the ring buffer becomes too full, the Main CPU must carefully increase VS1053b's playback samplerate as explained in Chapter 5.3, *Playback Speed Adjustment*. Similarly, if the buffer is too empty, playback samplerate is slightly decreased. If the changes are done slowly enough and with small enough steps, the listener can't notice them.

Note that VS1053b may encode at a slightly different speed from what is the nominal samplerate as explained in Chapter 2.6. Also, clock skew and other factors may affect the real speeds of the Server and Client(s). Thus, playback speed adjustment is a must in any streaming system. You may use a sample counter, presented in Chapter 5.1, to help control the end-to-end delay.

Note that Ogg Vorbis files are not designed to be resilient to bit errors. If you have a transmission channel that may lead to bit error (e.g. RF link), you should check for them in your transmission path and remove the packets that are broken. This will lead to unsynchronized audio until your playback speed adjustment corrects for that.

For further information, see Chapter 5.4, *Streaming Example*.

5.1 Sample Counter

An 32-bit sample counter, added for v1.3, is designed to help streaming Ogg Vorbis files. It tells the absolute sample number that is currently recorded through the ADC when an Ogg Vorbis file is being recorded. The sample counter is located at the beginning of the X memory user area, at address 0x1800, and it can be read through SCI.

The VS1053b Patches package has a similar counter. This makes it possible to maintain synchronization between the encoder and the decoder within an accuracy of a few samples if the data link has a constant delay and the sample counter data is transmitted from the sender to the receiver (e.g. RF link).

Note: The sample counter is not valid for the first few milliseconds of encoding.

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);
```

The VS1053b Patches package, which contains the other half of the Ogg Vorbis sample counter as well as some Ogg Vorbis decoding quality patches, is available at

<http://www.vlsi.fi/en/support/software/vs10xxpatches.html>

5.2 End-to-End Delay of a Streaming System

For a rough end-to-end delay estimate of a streaming Ogg Vorbis system, the following formula can be used:

$$d = \frac{4096 \times n_f \times b + 4000}{f_s}$$

where

- d is the result in seconds.
- n_f is bits 7:4 of SCI_AICTRL3, see Chapter 2.3.1. For this formula to work, you must set the bits to a non-zero value.
- $b = 1$ if you do byte-aligned read operations as instructed in Chapter 2.3.3. Otherwise $b = 2$.
- f_s is the samplerate.

Example: A 16 kHz voice profile is used. SCI_AICTRL3(7:4) is set to 1 for minimum delay. However, byte-aligned reads are not done. The system delay is:

$$d = \frac{4096 \times 1 \times 2 + 4000}{16000} = 0.76 \text{ s}$$

Naturally, you still have to add your transmission link delay to this value.

5.3 Playback Speed Adjustment

VS1053b's playback samplerate may be adjusted with the recommended *VS1053b Patches w/ FLAC Decoder* package, downloadable from <http://www.vlsi.fi/en/support/software/vs10xxpatches.html>.

The patch lets the user to adjust samplerate with approximately 2 PPM increments. For details, read the documentation for the VS1053b Patches package, particularly Chapter *Sample Rate Finetuning*.

Example:

you are encoding Ogg Vorbis data using XTALI=12.288 MHz at a nominal sample rate of 44100 Hz. However, you have learned from Chapter 2.6, *Samplerate Considerations*, the actual samplerate to be approximately 43885.7 Hz. How much do you need to adjust the samplerate?

$$\left(\frac{43885.7 \text{ Hz}}{44100 \text{ Hz}} - 1 \right) \times 512000 = -2488$$

Thus, write -2488 to the VS1053b Patches package Sample Rate Finetuning adjustment register.

Note: Adjusting the sample rate is a relatively expensive operation and it should not be done more often than once every 10 ms, preferably even less often than that.

5.4 Streaming Example

Let's take a streaming system that uses two VS1053b's, both running at 12.288 MHz crystals, and using best possible quality: the 44.1 kHz stereo Music Profile, Quality 5.

This is what you need to do on the encoder side:

- Load and start the VS1053b Ogg Vorbis Encoder.
- If you want to maintain synchronization for an indefinite time, send also the sample counter in your message packets. For how to read the sample counter, see Chapter 5.1, *Sample Counter*.

This is what you need to do on the decoder side:

- Load and run the VS1053b Patches package (with or without FLAC decoding) to the decoding VS1053b.
- As seen in Chapter 2.6, *Samplerate Considerations*, the encoding VS1053b records at 43885.7 Hz although it reports using 44100 Hz. So, set the playback rate as told in Chapter 5.3, *Playback Speed Adjustment*.
- If you want to maintain sync indefinitely, compare the recording sample counter value to the sample counter of the decoder. For how to read the sample counter at the decoder and, refer to the documentation of the VS1053b Patches package, Chapter *Sample Counter*.
- Decide what your target difference between the sample counters are. If, for instance, you are using Internet as your transfer medium, and you have lots of buffering in your microcontroller, the time may be e.g. 10 seconds (441000 samples). On an RF link, the time difference may be e.g. 0.5 seconds (22050 samples).
- By making very small positive and negative adjustments to the playback sample-rate, keep the recording and playback sample counter difference as constant as possible.

If you take all these steps, you will have a streaming system that both keeps synchronized for extended periods of time, and will self-adjust back to sync even after such issues as packet loss.

6 Latest Version Changes

This chapter describes the most important changes to the Vorbis VS1053b Ogg Vorbis Encoder and this document.

Version 1.70c, 2012-11-16

This is a documentation change only. Software is unchanged since v1.70.

- Added a Chapter 5.4, *Streaming Example*.
- Corrected values in Chapter 5.3, *Playback Speed Adjustment*, to correspond with values required by the VS1053b Patches package.

Version 1.70b, 2011-06-29

This is a documentation change only. Software is unchanged since v1.70.

- Changed recommendations for how to load and start the code in Chapter 2.3.2.
- Provided example for how to run the encoder in file `vs1053oggrec.c`. The file has also been quoted in Chapter 2.4.5.
- Added files in the binary `.img` image file format in the `profiles` directory. Documentation for the format also added to Chapter 4.2.

Version 1.70, 2010-09-09

This update adds functionality and enhances documentation.

- Added Vox operation for mono profiles. Other profiles are unchanged except for the bitstream version number. See Chapter 2.5 for details.
- Presented a rough estimate of end-to-end audio delay in Chapter 5.2.

7 Contact Information

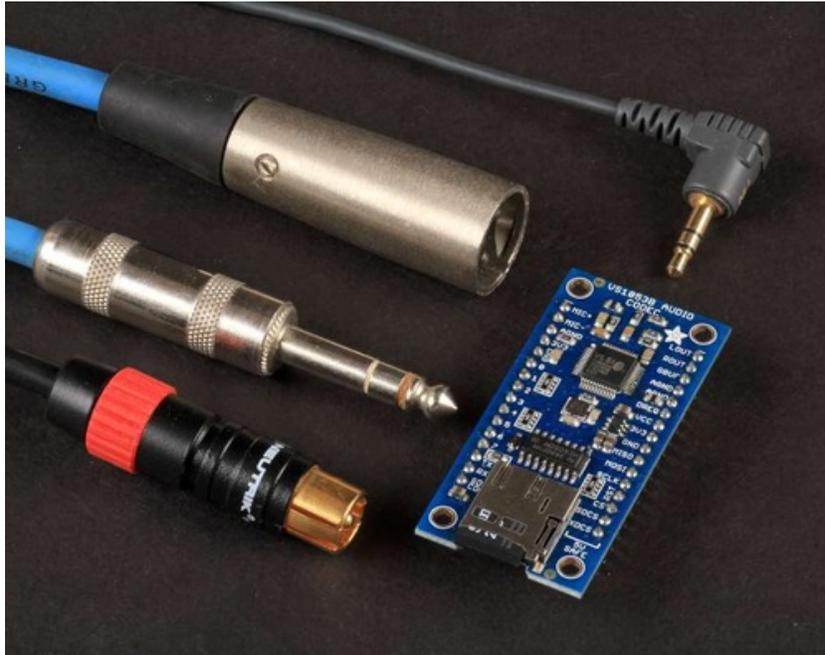
VLSI Solution Oy
Entrance G, 2nd floor
Hermiankatu 8
FI-33720 Tampere
FINLAND

Fax: +358-3-3140-8288
Phone: +358-3-3140-8200
Email: sales@vlsi.fi
URL: <http://www.vlsi.fi/>

For technical questions or suggestions regarding this application, please contact support@vlsi.fi.

Adafruit VS1053 MP3/AAC/Ogg/MIDI/WAV Codec Breakout Tutorial

Created by Bill Earl



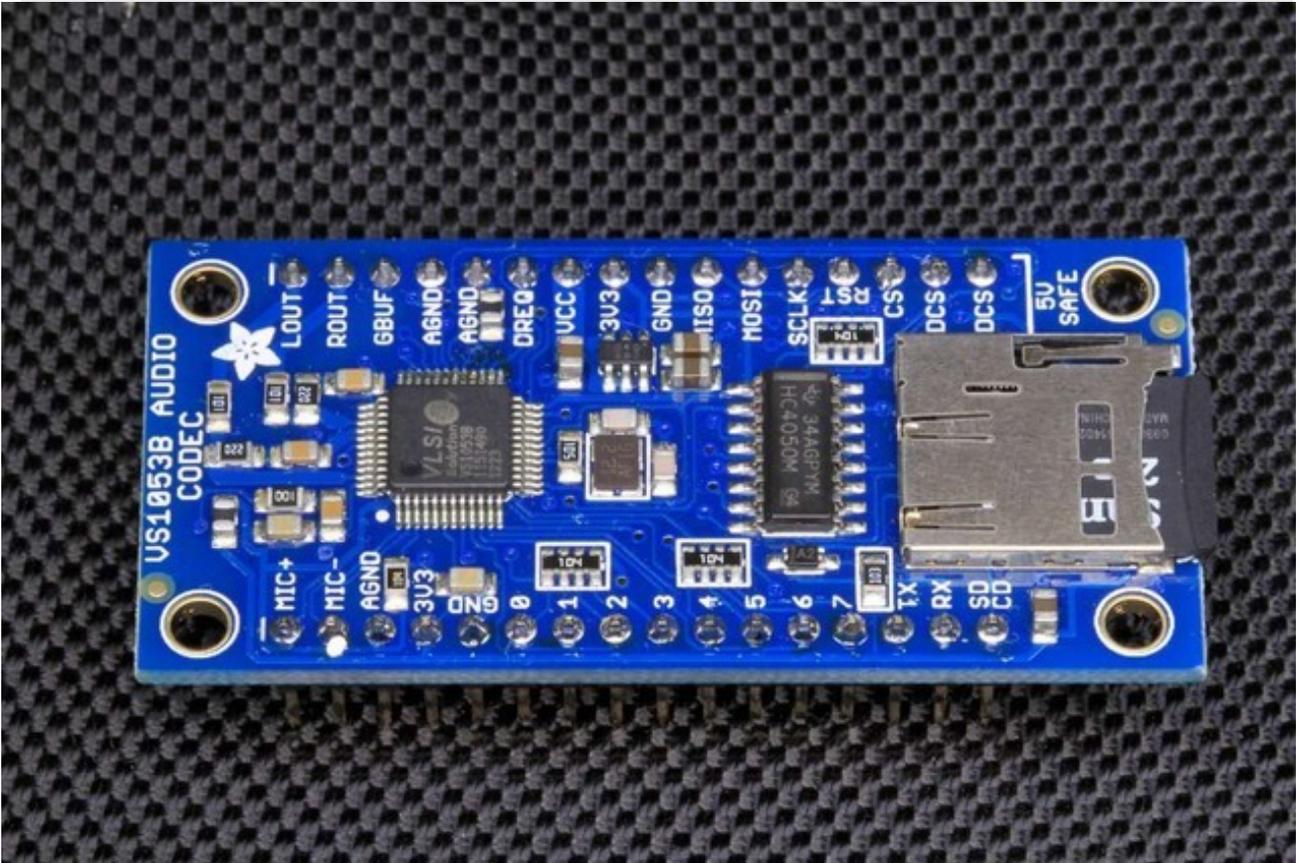
Last updated on 2014-05-16 04:15:17 PM EDT

Guide Contents

| | |
|---|----|
| Guide Contents | 2 |
| Overview | 4 |
| Assembly | 7 |
| Prepare the Headers | 8 |
| Position the Board | 8 |
| And Solder! | 9 |
| Getting Started - VS1053 Software | 10 |
| Simple Audio Player Wiring | 11 |
| Prepare the breadboard | 12 |
| Connect SPI and Reset | 12 |
| Connect the rest of the digital signals | 12 |
| Connect the Headphone Jack | 13 |
| Load some MP3 files | 13 |
| Simple Audio Player Sketch | 14 |
| For the Micro: | 14 |
| Audio Connections | 16 |
| MIDI Connections | 19 |
| Prepare the breadboard | 19 |
| Configure for MIDI operation | 19 |
| Connect the Headphone Jack | 19 |
| Midi Example Sketch | 20 |
| GPIO | 21 |
| Start with the basic wiring | 21 |
| Add some LEDs | 21 |
| Run the player_gpiotest sketch | 21 |
| Ogg Recorder | 23 |
| Recording with the VS1053 | 23 |
| Wiring | 24 |
| Basic SPI connections: | 24 |

| | |
|-----------------------------------|----|
| Additional Control Signals: | 24 |
| Start/Stop Button (Momentary): | 24 |
| Electret Microphone Circuit: | 25 |
| A note about microphone circuits: | 26 |
| Recording Sketch: | 27 |
| Plug-Ins | 27 |
| To Record: | 27 |
| To Playback: | 27 |
| Library Reference | 28 |
| class Adafruit_VS1053_FilePlayer | 28 |
| Public Methods: | 28 |
| Public Member Variables: | 28 |
| class Adafruit_VS1053 | 28 |
| public Methods: | 28 |
| Downloads and Links | 31 |
| Library: | 31 |
| Technical Information: | 31 |
| Schematics for v2 | 31 |

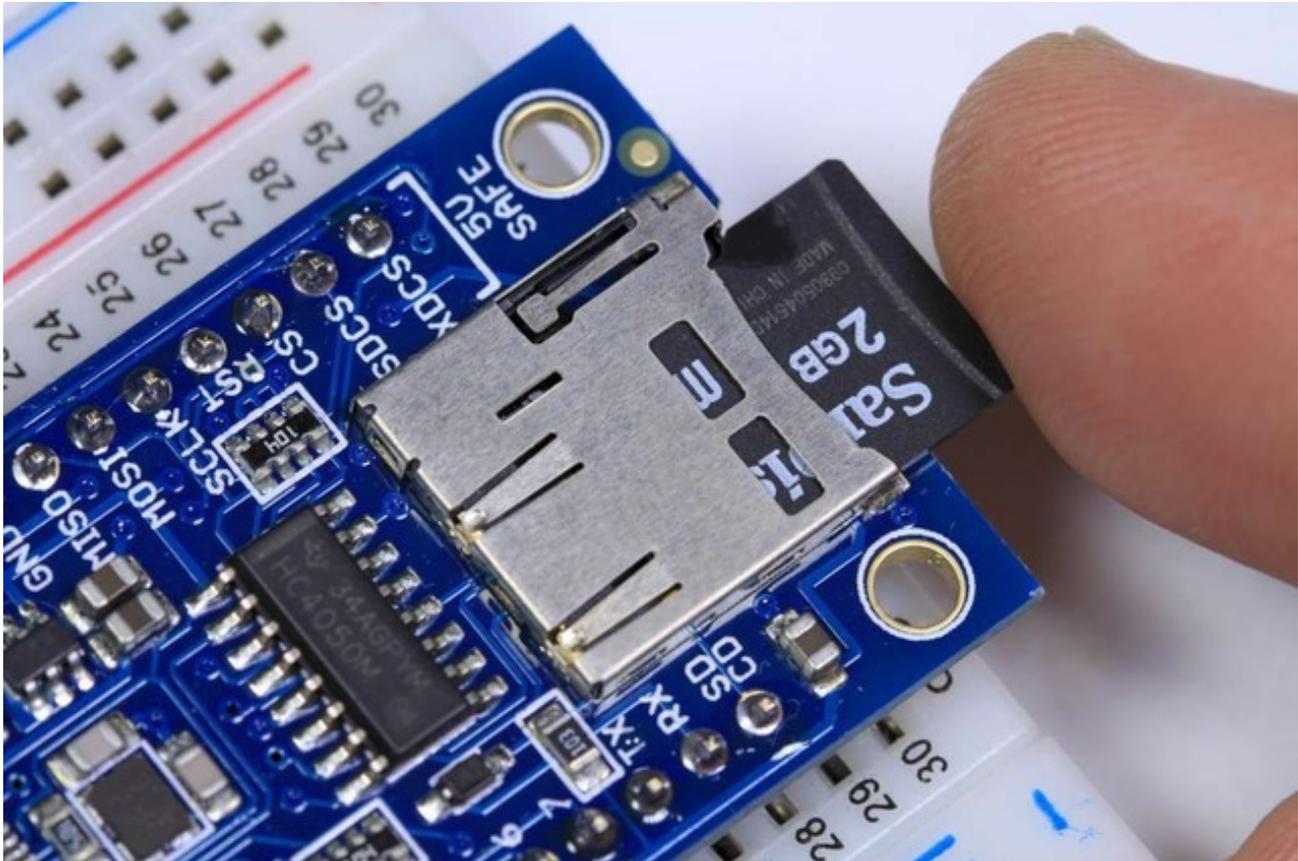
Overview



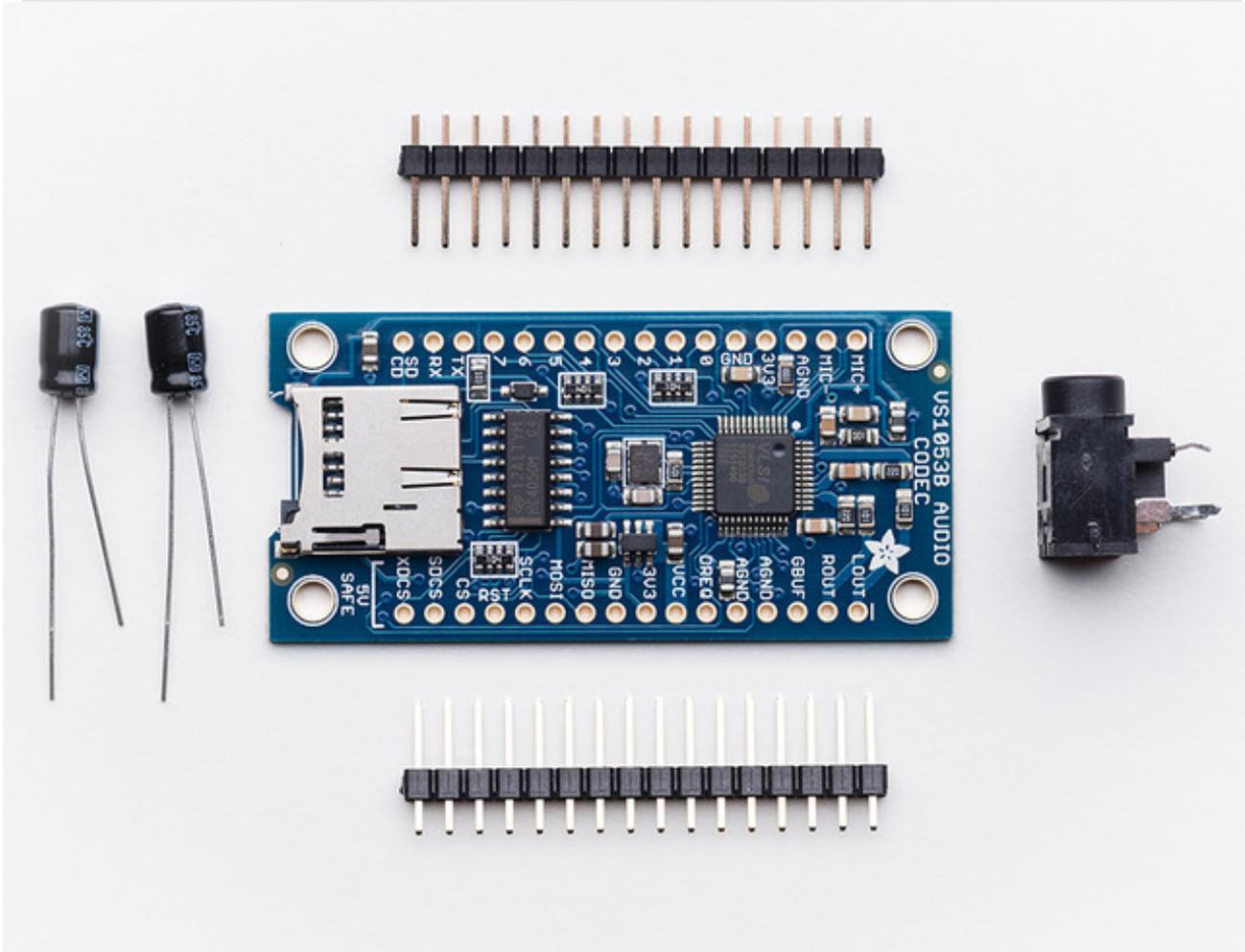
Above: v1.0 codec board. Below: v2.0 codec board

connection. You'll likely want to play music from a microSD card so we added a holder on-board. And since we know so many of our customers use 5V microcontrollers like the Arduino, we made the interface pins all 5V compliant with level shifters so you can use the chip at 3V or 5V power/logic!

The breakout board comes fully assembled and tested. All v1.0 boards come with two 100uF electrolytic capacitors for line-level output coupling, v2.0 have the capacitors integrated into the PCB. All kits come with some 0.1" male header you can solder to the breakout so it plugs into a breadboard and a bonus stereo headphone jack that will be handy when you want to plug headphones in!

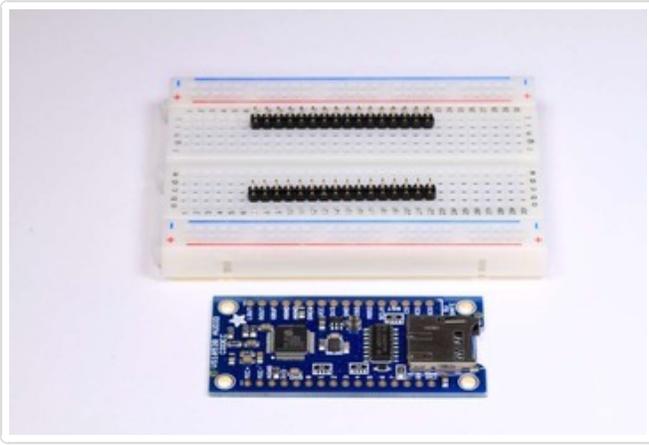


Assembly



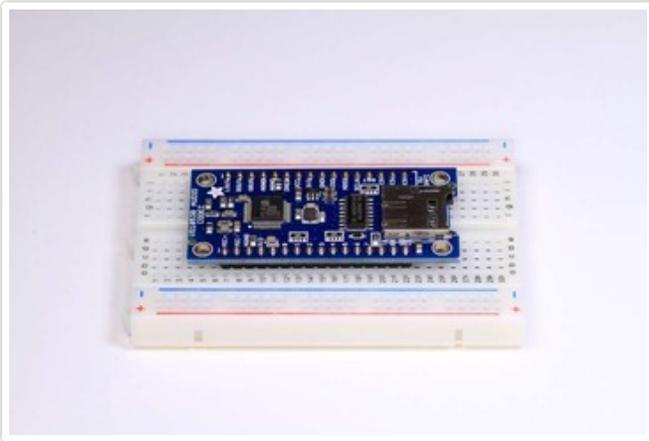
The board comes with all surface-mount components assembled and tested. For use on a breadboard, you will want to install the included header strips. This is a simple process and should only take a few minutes.

Assembly is the same for both v1 and v2 breakouts, even if the boards look a bit different!



Prepare the Headers

Cut the header strips to length if necessary and place the long-pins-down in the breadboard.

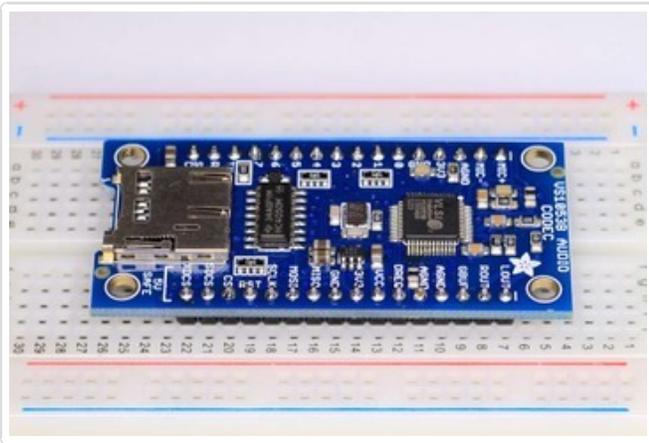
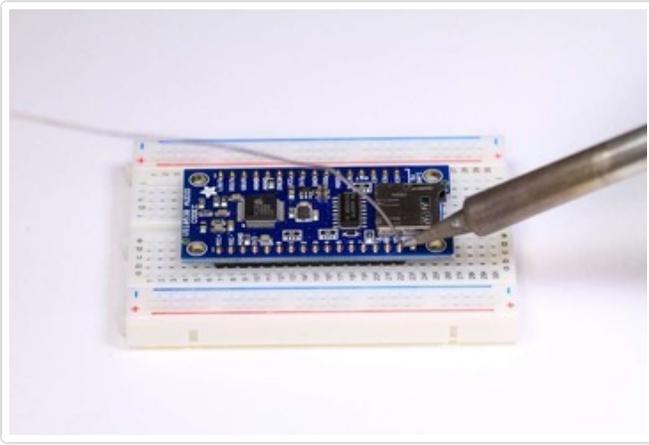


Position the Board

Place the breakout board over the header pins.

And Solder!

Solder every pin to assure good electrical contact.



Getting Started - VS1053 Software

To get started with the VS1053 breakout (any version) you will first need to download the Adafruit VS1053 Library:

Download Adafruit VS1053
Library

<http://adafru.it/cDQ>

Copy the folder inside the zip file to the Libraries folder inside your Arduino Sketchbook folder and re-name it to Adafruit_VS1053 For more details on how to install Arduino libraries, check out our detailed tutorial using the link below:

All About Arduino Libraries

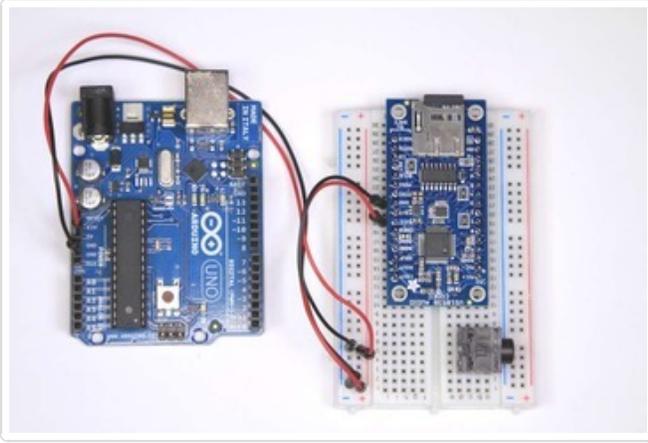
<http://adafru.it/aYM>

Simple Audio Player Wiring



The VS1053 can be configured as a simple audio player under control of the Arduino. The Arduino reads data from the on-board SD card, then plays it back through the CODEC via the SPI interface.

These steps are identical for both v1 and v2 codec breakouts, even if the boards look a little bit different

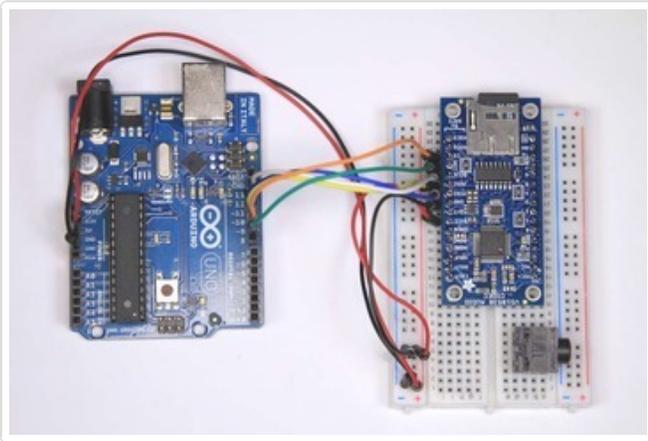


Prepare the breadboard

Place the VS1053 breakout on the breadboard. Center it so that there is one row of holes on each side.

Add the breakout friendly headphone jack and wire power and ground jumpers as shown.

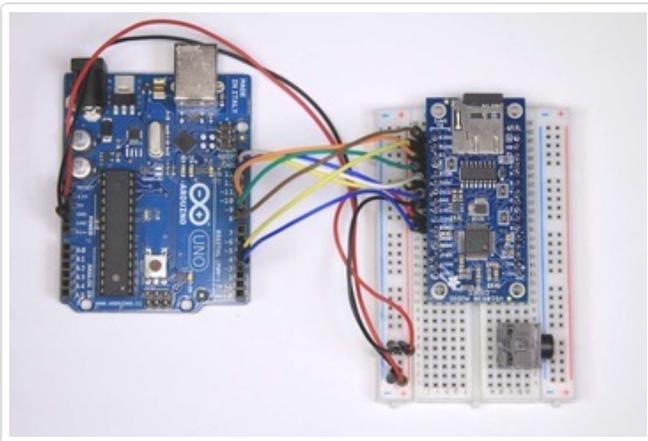
- **VCC -> 5v**
- **GND -> GND**



Connect SPI and Reset

Add jumpers for:

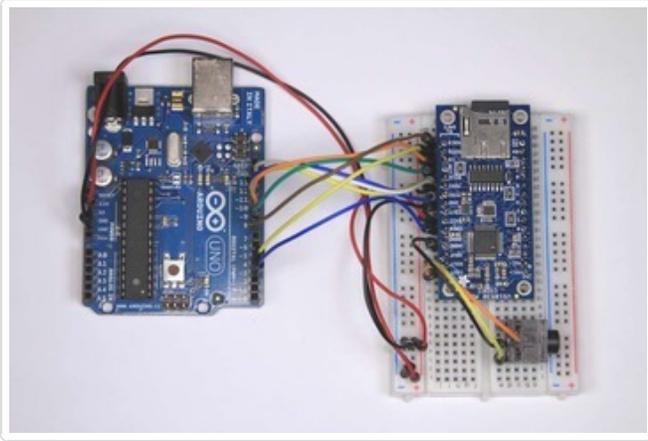
- **CLK -> Arduino #13 (Mega 52)**
- **MISO -> Arduino #12 (Mega 50)**
- **MOSI -> Arduino #11 (Mega 51)**
- **CS -> Arduino #10**
- **RST -> Arduino #9**



Connect the rest of the digital signals

Add more jumpers for:

- **XDCS -> Arduino #8**
- **SDCS -> Arduino #4**
- **DREQ -> Arduino #3**



Connect the Headphone Jack

This step will be a little bit different depending on which version you have. If your breakout is v2 and has the round silver capacitors, add jumpers for:

- **AGND** -> **Center 'ground' Pin**
- **LOUT** -> **Left Pin**
- **ROUT** -> **Right Pin**

(these are in the corner in a row)

If your breakout is v1, the pins will be labeled

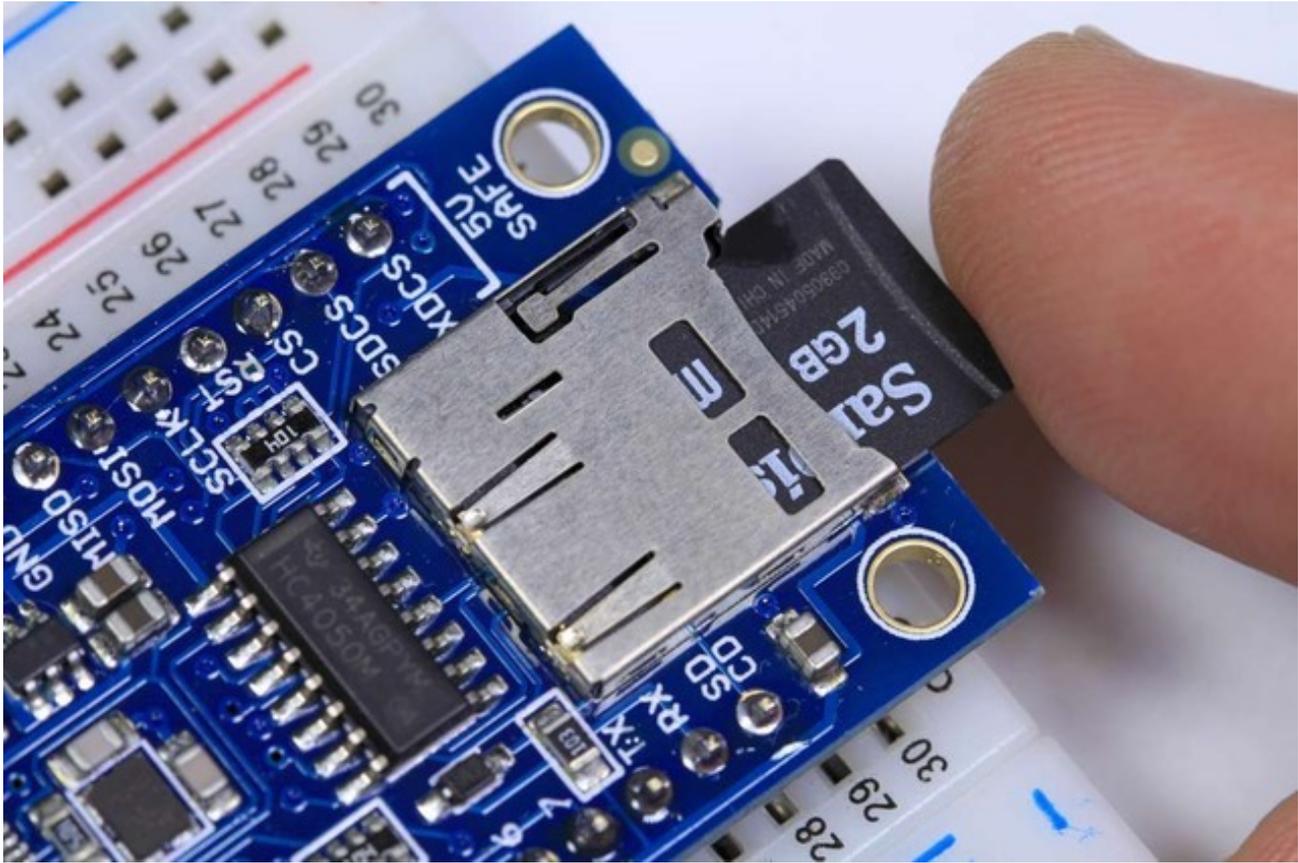
- **GBUF** -> **Center 'ground' Pin**
- **LOUT** -> **Left Pin**
- **ROUT** -> **Right Pin**

For v1, do not use this wiring for powered speakers or a line-in connection to a computer/stereo. **Only for headphones!**

For v2, you can use this wiring for any type of connection without worry.

Load some MP3 files

Copy 2 MP3 files to a micro SD card and name them **track001.mp3** and **track002.mp3** (this is just for the test, you can re-name them later). Then install the card in the slot on the VS1053 breakout.



Make sure you have a good quality SD card, some cheap SD cards won't work, causing confusion! Especially 'non-brand' knockoffs.

Simple Audio Player Sketch

Connect the Arduino to your computer with a USB cable and plug your headphones into the headphone jack. Select **File->Examples->Adafruit_VS1053_Codec->player_simple** to load the example code.

You should hear your MP3 files in the headphones.

You can also run **File->Examples->Adafruit_VS1053_Codec->player_interrupts**. This example demonstrates playing files in the background using interrupts. This allows you to do other things in your sketch while the music plays!

With both examples, you can change the file names in the code to play different files - or even extend the code to search the SD card for files to play.

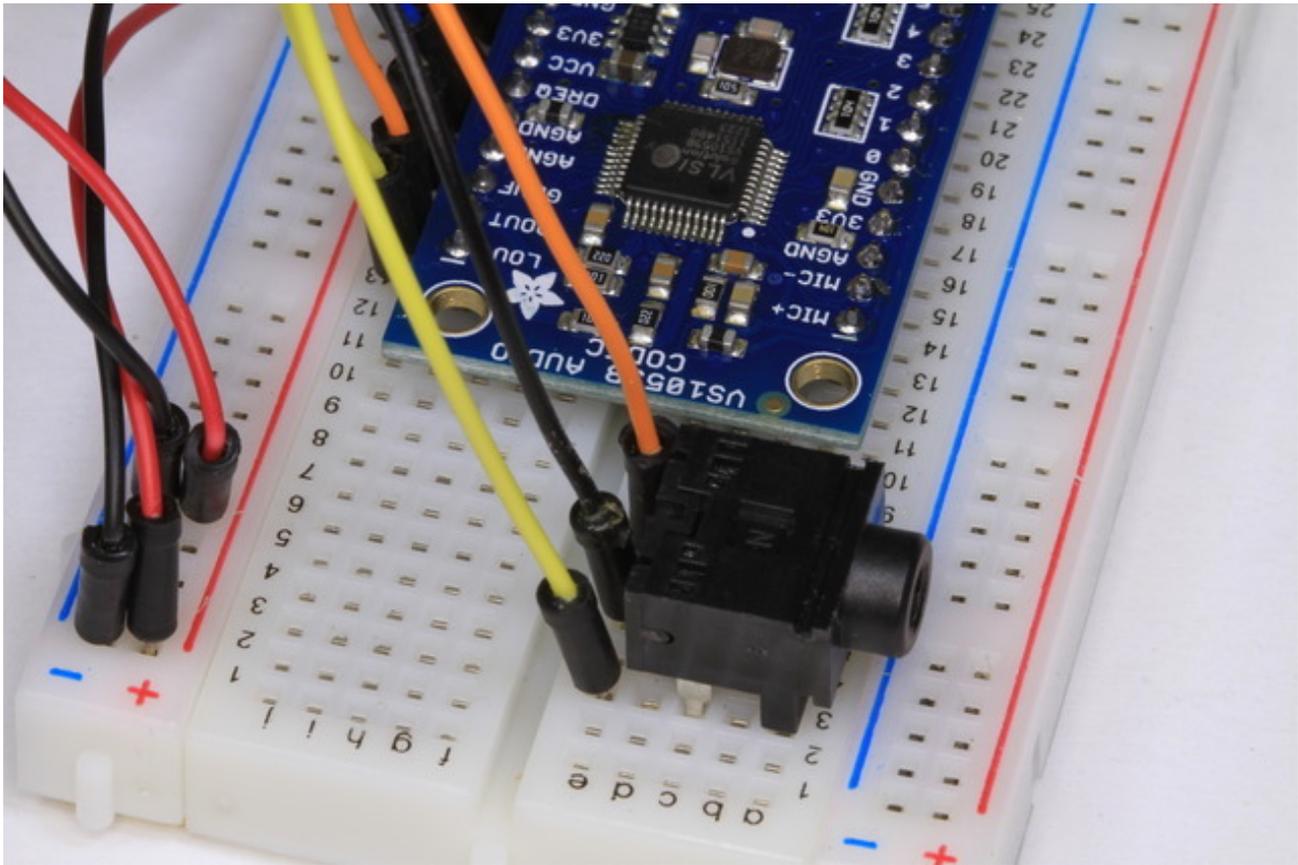
For the Micro:

Frank Cohen has submitted a wiring diagram for the FilePlayer example on a Micro. See his code posted here as well: <http://votsh.files.wordpress.com/2014/02/vs1053-arduino-micro-connections.pdf>

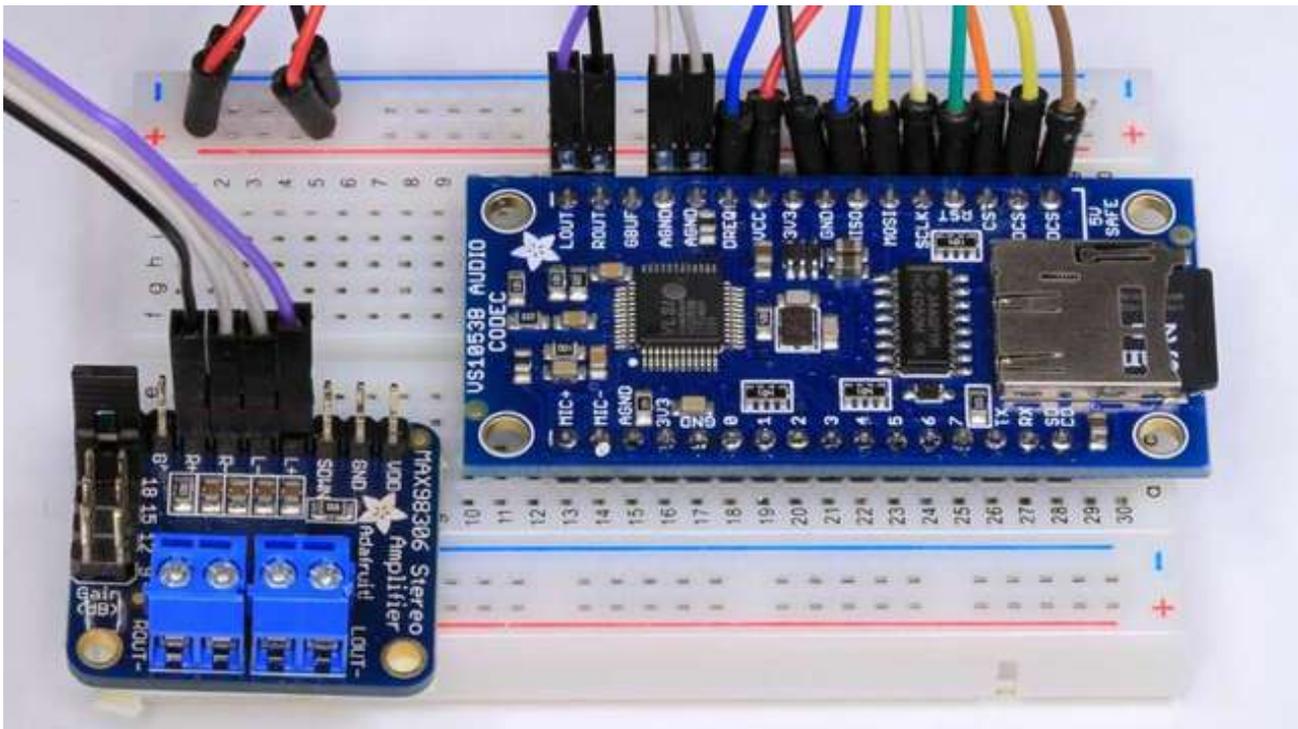
Audio Connections



Both the VS1053 version 1 (shown above, does not have big round silver capacitors at the top) **and** the version 2 (has a v2 label, has two big round silver caps) can drive a pair of headphones directly. Just connect the LOUT and ROUT to the left and right pins of the headphone jack. The center pin should be connected to GBUF.



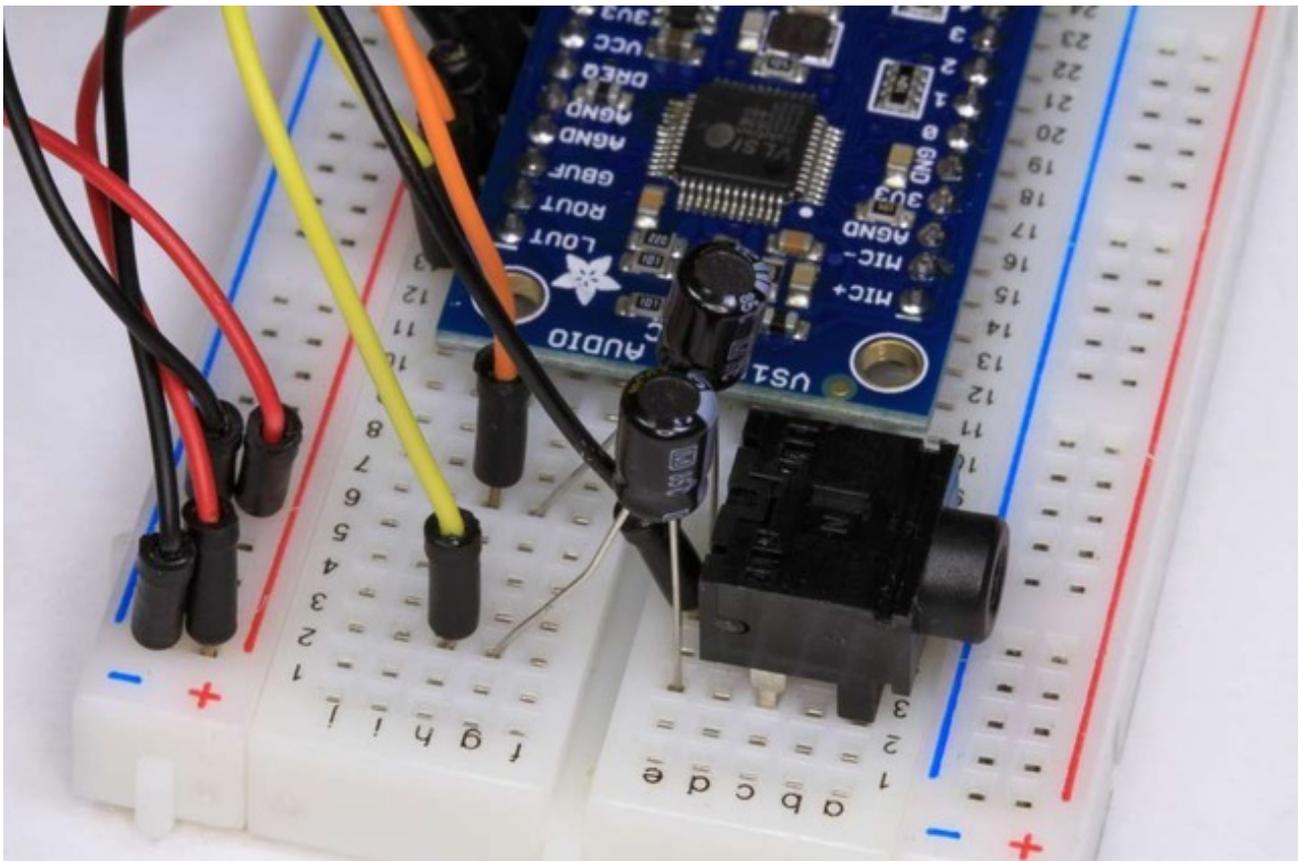
Both v1 and v2 can also directly drive audio devices with differential inputs such as our 3.7 Watt Stereo Class D Amplifier. Connect LOUT and ROUT to the L+ and R+ pins of the amplifier. Connect L- and R- to the AGND pins.



If you have a v1 breakout, you can connect to powered speakers/stereo BUT YOU MUST USE THE INCLUDED CAPACITORS!

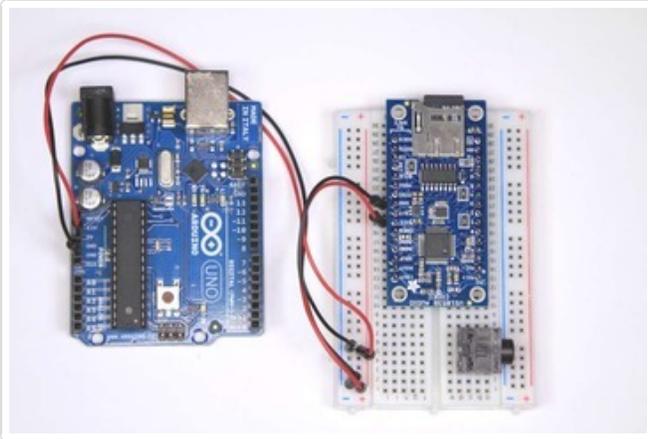
For audio devices requiring a 'line level' input, use the included capacitors. These are polarized electrolytic capacitors, so it matters which way you connect them. Connect the positive (longer) leads to the LOUT and ROUT pins, and the negative leads to the L & R pins of the headphone jack or L and R inputs of the audio device. The common (center) pin of the audio jack should be connected to the AGND pin.

If you have a v2 breakout, these capacitors can be skipped, they are included onto the breakout board itself.



MIDI Connections

With a few jumper connections, the board will boot up in MIDI mode that will read 'classic' 31250Kbaud MIDI data on a UART pin and act like a synth/drum machine - there are dozens of built-in drum and sample effects.

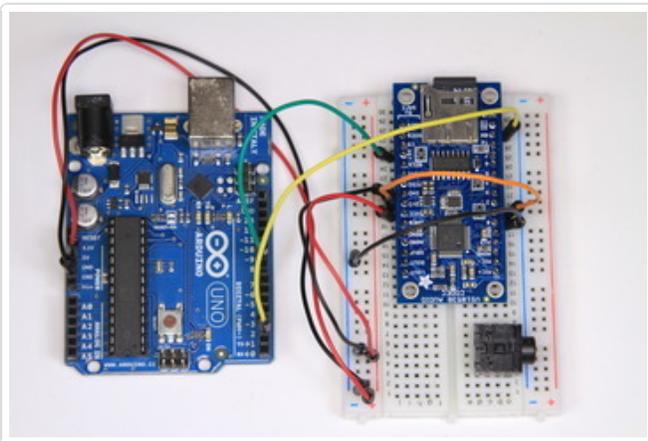


Prepare the breadboard

Place the VS1053 breakout on the breadboard. Center it so that there is one row of holes on each side.

Add the breakout friendly headphone jack and wire power and ground jumpers as shown.

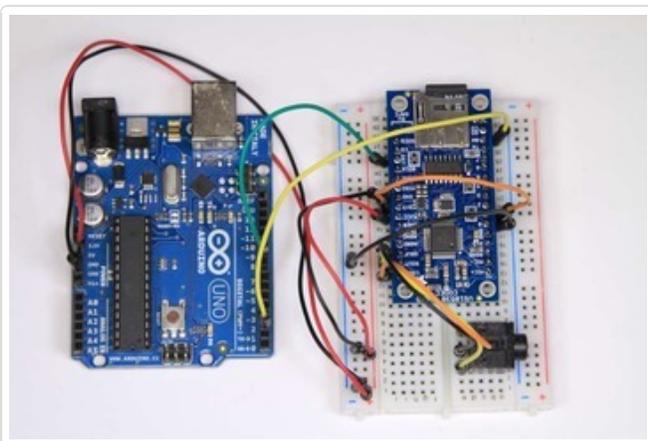
- **VCC -> 5v**
- **GND -> GND**



Configure for MIDI operation

Add jumpers as follows:

- **GPIO-0 -> GND**
- **GPIO-1 -> 3.3v**
- **Rx -> Arduino Digital #2**
- **RST -> Arduino Digital #9**



Connect the Headphone Jack

Same as the earlier tests, add jumpers for:

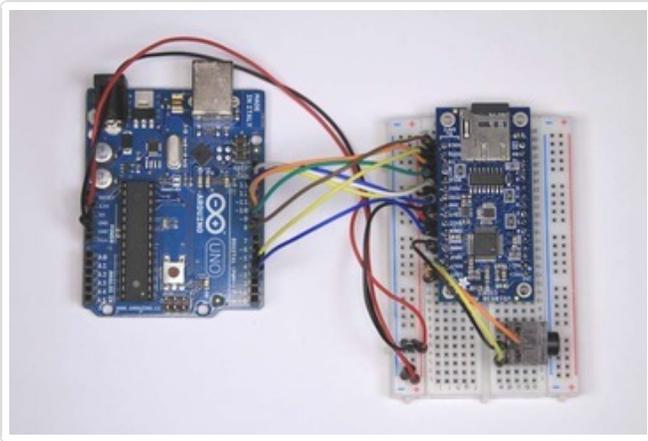
- **AGND (v2) or GBUF (v1) -> Center Pin**
- **LOUT -> Left Pin**
- **ROUT -> Right Pin**

Midi Example Sketch

Connect the Arduino to your computer with a USB cable and plug your headphones into the headphone jack. Select **File->Examples->Adafruit_VS1053_Codec->player_miditest** to load the example code for MIDI operation.. You should hear a repeating series of ascending tones from the MIDI player.

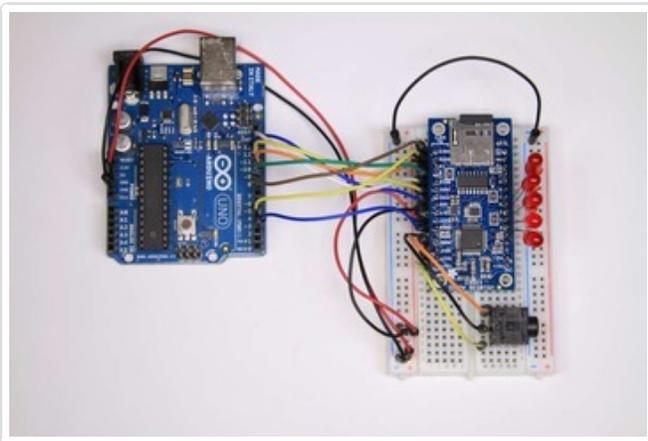
GPIO

The VS1053 has 8 GPIO pins that can be read and written via the library. The `player_gpiotest` sketch demonstrates how to do this.



Start with the basic wiring

Use the Simple Audio Player Wiring from the link on the left.



Add some LEDs

Add a jumper from the ground bus on the left to the ground bus on the right of the breadboard.

Then add some LEDs from the GPIO pins to ground. Be sure to connect the long lead (anode) to the GPIO pin and the short lead (cathode) to ground.

What? No current limiting resistors?

Strictly speaking, best practice is to use a current limiting resistor when driving an LED from a GPIO pin. In this case, the example sketch pulses each led only briefly, so there is no danger of damage. For more general use, you should select a resistor appropriate for the led you are using. See [All About LEDs \(http://adafru.it/cIH\)](http://adafru.it/cIH) for more detail.

Run the `player_gpiotest` sketch

Connect the Arduino to your computer with a USB cable. Select **File->Examples->Adafruit_VS1053->player_gpiotest** to load the example code.

If you have headphones, you will hear a beep at the start to indicate that the sketch is running. Then you should see the LEDs flashed in sequence.

If you open the Serial Monitor, you can see the values that are written to and read from each GPIO pin.

Note that the GPIO pins are NOT 5v safe. Do not connect them to voltages greater than 3.3v.

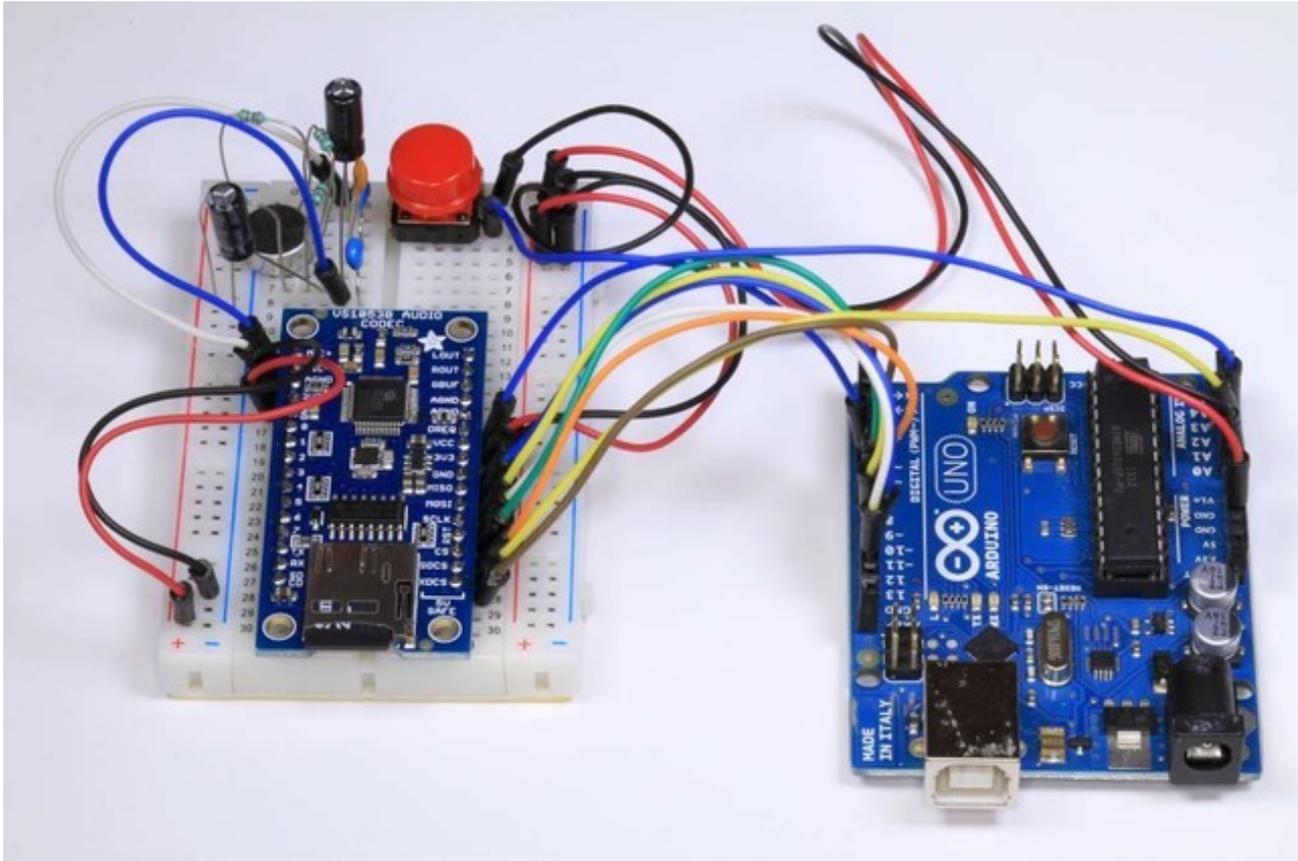
Ogg Recorder



(Nipper photo by Beverly & Pack - Creative Commons Share Alike)

Recording with the VS1053

The `record_ogg` example sketch turns your VS1053 breakout into a recording device that generates OGG encoded files on the SD card in real-time.



Wiring

Wiring for the record_ogg sketch is as follows:

Power and Ground:

- VCC -> 5v
- GND -> GND

Basic SPI connections:

- SCLK -> Arduino #13
- MOSI -> Arduino #11
- MISO -> Arduino #12
- CS -> Arduino #10

Additional Control Signals:

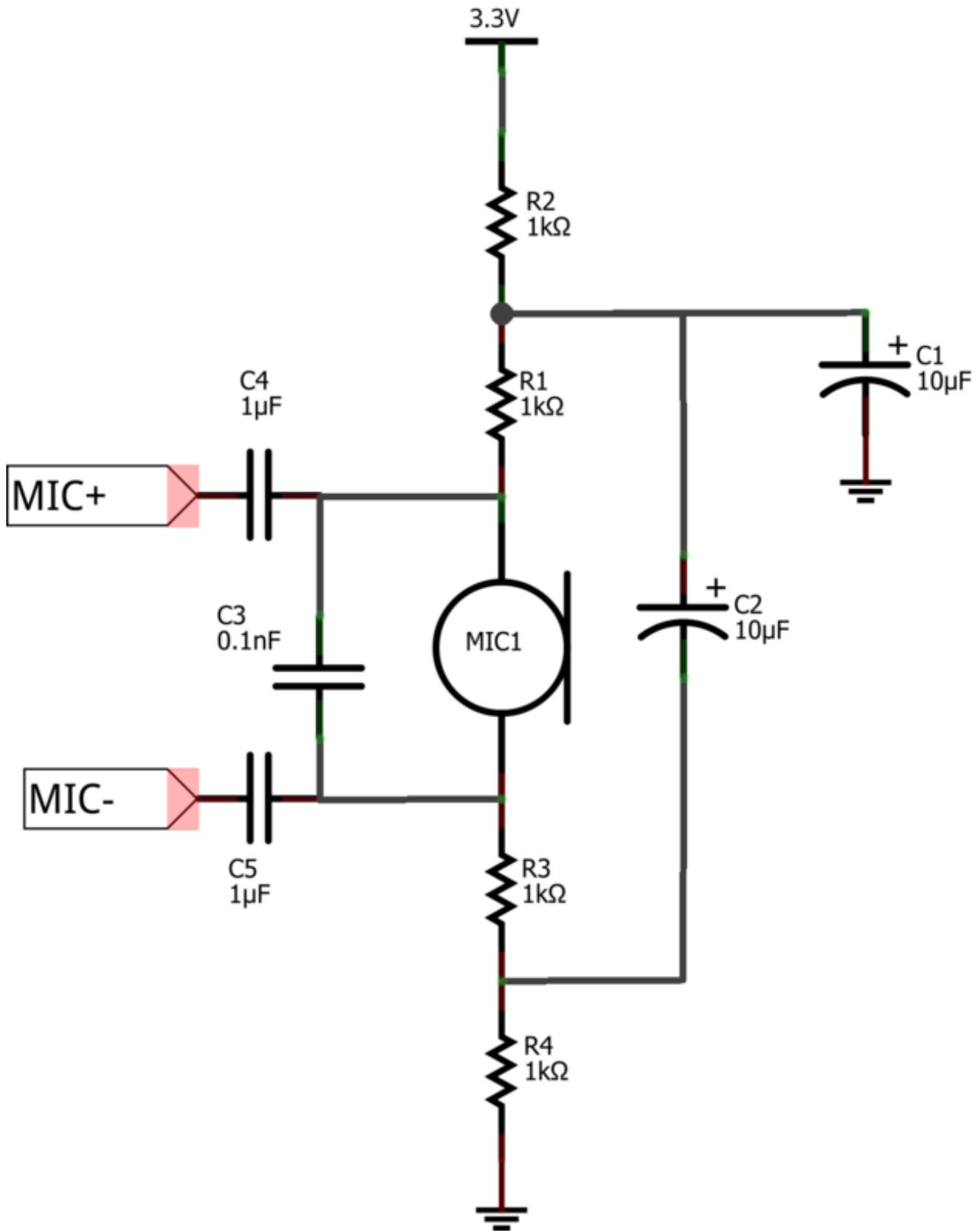
- SDCS -> Arduino A0
- XDCCS -> Arduino #8
- RST -> Arduino #9
- DREQ -> Arduino A1

Start/Stop Button (Momentary):

- from Arduino #7 to GND

Electret Microphone Circuit:

The Microphone circuit is derived from the reference design shown in Figure 3 on Page 13 of the [VS1053B Datasheet](http://adafru.it/c1l) (<http://adafru.it/c1l>). Note that the ground connections shown are to analog ground (AGND).



Made with  Fritzing.org

A note about microphone circuits:

Microphones, by their nature, are very sensitive devices. They are prone to picking up all sorts of noises (both electrical and acoustic) in addition to what you are trying to record. A microphone circuit built on a breadboard is not likely to sound good at all. For good sound, you will want to review the spec sheet recommendations and build this on a good quality circuit board.

Recording Sketch:

Connect the Arduino to your computer with a USB cable and plug your headphones into the headphone jack. Select **File->Examples->Adafruit_VS1053_Codec->record_ogg** to load the example code.

Plug-Ins

The record_ogg example sketch demonstrates a powerful feature of the VS1053 breakout: Plug-ins. OGG recording capability is not supported natively in the chip itself, but through a plug-in code module. A binary file containing the code image of the plug-in is loaded from the SD card at startup. Once loaded, the VS1053 becomes an OGG recording device.

To Record:

Press the start/stop button to start the recording. Press once again to stop. Each recording will be stored in a separate .OGG file on the SD card.

To Playback:

Use any OGG capable audio player on your computer - or playback through the VS1053 using the Simple Audio Player configuration (see link at left).

Library Reference

class Adafruit_VS1053_FilePlayer

The `Adafruit_VS1053_FilePlayer` class is derived from the `Adafruit_VS1053` class and provides high level functions for playing files stored on the VS1053 breakout SC Card reader.

Public Methods:

Adafruit_VS1053_FilePlayer (uint8_t mosi, uint8_t miso, uint8_t clk, uint8_t rst, uint8_t cs, uint8_t dcs, uint8_t dreq, uint8_t cardCS) - Software SPI constructor. Uses Software SPI, so you must specify all SPI pins.

Adafruit_VS1053_FilePlayer (uint8_t rst, uint8_t cs, uint8_t dcs, uint8_t dreq, uint8_t cardCS) - Hardware SPI constructor. Uses Hardware SPI and assumes the default SPI pins.

boolean begin(void) - Initialize communication and reset the chip.

boolean useInterrupt(uint8_t type) - Specifies the interrupt to use for interrupt-driven playback. Valid arguments are:

- `VS1053_FILEPLAYER_TIMER0_INT`
- `VS1053_FILEPLAYER_PIN_INT`

boolean startPlayingFile(char *trackname) - Begin playing the specified file from the SD card using interrupt-driven playback. This allows your program to perform other tasks as the file is playing.

boolean playFullFile(char *trackname) - Play the complete file. This function will not return until the playback is complete.

Public Member Variables:

File currentTrack - File currently being played

boolean playingMusic - True if playback in progress

class Adafruit_VS1053

The `Adafruit_VS1053` class implements an interface to the basic VS1053 functionality. For more detail on the operation of the VS1053 chip, please refer to the documentation on the Downloads page (see the link to the left).

public Methods:

Adafruit_VS1053(uint8_t mosi, uint8_t miso, uint8_t clk, uint8_t rst, uint8_t cs, uint8_t dcs, uint8_t dreq) - Software SPI constructor - must specify all pins.

Adafruit_VS1053(uint8_t rst, uint8_t cs, uint8_t dcs, uint8_t dreq) - Hardware SPI constructor - assumes hardware SPI pins.

uint8_t begin(void) - Initialize SPI communication and (hard) reset the chip.

void reset(void) - Performs a hard reset of the chip.

void softReset(void) - Attempts a soft reset of the chip.

uint16_t sciRead(uint8_t addr) - Reads from the specified register on the chip.

void sciWrite(uint8_t addr, uint16_t data) - Writes to the specified register on the chip.

void sineTest(uint8_t n, uint16_t ms) - Generate a sine-wave test signal.

void spiwrite(uint8_t d) - Low-level SPI write operation.

uint8_t spiread(void) - Low-level SPI read operation.

uint16_t decodeTime(void) - Reads the DECODETIME register from the chip.

void setVolume(uint8_t left, uint8_t right) - Set the output volume for the chip.

void dumpRegs(void) - Prints the contents of the MODE, STATUS, CLOCKF and VOLUME registers.

void playData(uint8_t *buffer, uint8_t buffersiz) - Decode and play the contents of the supplied buffer.

boolean readyForData(void) - Test if ready for more data.

void applyPatch(const uint16_t *patch, uint16_t patchsize) - Apply a code patch (See datasheet for details).

uint16_t loadPlugin(char *fn) - Load the specified plug-in.

void GPIO_digitalWrite(uint8_t i, uint8_t val) - Write to a GPIO pin.

void GPIO_digitalWrite(uint8_t i) - Write to all 8 GPIO pins at once.

uint16_t GPIO_digitalRead(void) - Read all 8 GPIO pins at once.

boolean GPIO_digitalRead(uint8_t i) - Read a single GPIO pin.

void GPIO_pinMode(uint8_t i, uint8_t dir) - Set the Pin Mode (INPUT/OUTPUT) for a GPIO pin.

boolean prepareRecordOgg(char *plugin) - Initialize chip for OGG recording.

void startRecordOgg(boolean mic) - Start recording (mic = true for microphone input).

void stopRecordOgg(void) - Stop the recording.

uint16_t recordedWordsWaiting(void) - Returns the number of words recorded.

uint16_t recordedReadWord(void) - Reads the next word from the buffer of recorded words.

uint16_t recordedReadWord(void) - Reads the next word from the buffer of recorded words.

Downloads and Links

Library:

- [Adafruit VS1053 Library \(http://adafru.it/cLE\)](http://adafru.it/cLE)

Technical Information:

- [VS1053B \(Codec chip\) datasheet \(http://adafru.it/cII\)](http://adafru.it/cII)
- [Details about the Ogg vorbis encoder/recorder \(http://adafru.it/cIJ\)](http://adafru.it/cIJ)
- [App note on connecting to analog outputs! \(http://adafru.it/cIK\)](http://adafru.it/cIK)
- Dimensions: 27.72mm / 1.09" x 58.08mm / 2.28"

Schematics for v2

