# VS1005 VSOS AUDIO SUBSYSTEM

VS1005g

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

| Revision History | | | |
|---|---|---|---|
| **Rev.** | **Date** | **Author** | **Description** |
| 3.60 | 2020-10-13 | HH | Updated for VSOS 3.60. |
| 3.58 | 2019-08-24 | HH | Added e.g. decape.dl3 to Chapter 13. |
| 3.57 | 2019-04-10 | HH | Minor corrections. |
| 3.55b | 2018-06-22 | HH | Added Noise Killer audio driver. |
| 3.55a | 2018-04-24 | HH | New features to Reverb Generator audio driver. |
| 3.55 | 2018-04-05 | HH | Added Reverb Generator audio driver. |
| 3.52 | 2018-01-22 | HH | Updated for VSOS 3.52. |
| 3.42 | 2017-05-18 | HH | More S/PDIF drivers, FLAC Encoder, Pitch Shifter. |
| 3.40 | 2016-11-03 | HH | Updated for VSOS 3.40. |
| 3.30a | 2016-07-14 | HH | Bug patch release for VSOS 3.30. |
| 3.30 | 2016-06-22 | HH | Updated for VSOS 3.30. |
| 1.03 | 2016-02-15 | HH | New AUXPLAY, fract. rates, updated AuOutput. |
| 1.02 | 2016-01-27 | HH | Added AuOutput app and Slave Sync drivers. |
| 1.01 | 2015-09-14 | HH | Corrections, e.g. Figure 8. |
| 1.00 | 2015-09-04 | HH | Initial release. |

# Contents

## List of Figures

# 1 Introduction

The VS1005 VSOS offers many, versatile audio drivers.

This document explains how to use the numerous drivers to your best advantage.

After the disclaimer and definitions in Chapters 2 and 3, an overview of the Audio subsystem is given in Chapter 4, *Overview*, followed by requirements in Chapter 5, *Requirements*.

The VSOS audio subsystem is presented in Chapter 6, *The VS1005 VSOS Audio Subsystem*.

The currently existing audio drivers are presented in Chapter 7, *Audio Drivers*, followed by a presentation of the currently existing filters in Chapter 8, *Audio Driver Filters*, and control programs in Chapter 9, *Audio Control Programs*,

Some examples on how to start audio drivers from config.txt or the VSOS Shell are shown in Chapter 10, *Configuration Examples*.

Chapter 12 shows how to control some aspects on audio using UiMessages, even if the program that is currently running doesn't have any audio controls.

Audio Decoders are presented in Chapter 13, and Audio Encoders in Chapter 14.

The document ends with Chapter 15, *Latest Document Version Changes*, and Chapter 16, *Contact Information*.

## 2 Disclaimer

VLSI Solution makes everything it can to make this documentation as accurate as possible. However, no warranties or guarantees are given for the correctness of this documentation.

## 3 Definitions

**DSP** Digital Signal Processor.

**I-mem** Instruction Memory.

**LSW** Least Significant (16-bit) Word.

**MSW** Most Significant (16-bit) Word.

**RISC** Reduced Instruction Set Computer.

**VS_DSP**[4] VLSI Solution's DSP core.

**VSIDE** VLSI Solution's Integrated Development Environment.

**VSOS** VLSI Solution's Operating System.

**X-mem** X Data Memory.

**Y-mem** Y Data Memory.

# 4 Overview

The VSOS Audio Subsystem provides numerous drivers to handle the many audio Input/Output options of VS1005. The audio drivers can be controlled either with ioctl() calls from the C language, or from VSOS Shell control program.

While instructions for how to use each audio driver are provided in the README.TXT or documentation .PDF files of the drivers, this document will provide an overview of the capabilities of the drivers. However, for details, refer to documentation of the audio drivers themselves.

# 5 Requirements

To test the audio drivers in this document, you need to have the following building blocks:

- VS1005g Developer Board. The VS1005g BreakOut Board should also work, but these instructions have been tested with the DevBoard.
- Latest version of VSOS installed (at least v3.23, released 2015-09-04).
- USB cable between DevBoard and PC for uploading new software.
- If you want to use the VSOS Shell environment, you will also need:
  - UART or USB->UART cable connected between DevBoard and PC for using the UART interface. Data speed is 115200 bps, format is 8N1.
  - Your favorite UART Terminal Emulation program installed on the PC. Read the “VS1005 VSOS Shell” for further details.

When all of this is in order, you are ready to test the VSOS Audio Subystem.

# 6 The VS1005 VSOS Audio Subsystem

As a default, VSOS offers a simple audio output driver that lets the user output 16-bit mono or stereo audio to the VS1005 analog output pins LEFT and RIGHT, and control the sample rate.

VSOS Audio makes it very easy to produce sound with its standard-C-like standard audio interface (Chapter 6.1, *Standard Audio*). Instead of being forced to use audio-specific I/O routines, audio looks just like files.

More complex audio operations and redirections can be done using the audio drivers, described Chapter 7, *Audio Drivers*.

## 6.1 Standard Audio

VSOS offers the user a standard audio source and destination, although the audio source is only activated if an appropriate audio input driver is loaded (Chapter 7). Called *stdaudioin* and *stdaudioout*, standard audio file handles are to sound much like *stdin* and *stdout* are to standard input and output in standard C. It is not allowed for the user to close standard audio input or output files, but the user may modify their parameters.

By default, *stdaudioout* is connected to analog output pins LEFT and RIGHT, although this can be changed with appropriate audio drivers.

Both standard audio input and output open in stereo, 16-bit, 48 kHz mode. These parameters can be changed by the user, with driver and hardware dependent limitations.

The user may use all standard read and write operations to read from and write to standard audio. It is, however, required that fread() / fwrite() functions are used instead of character-based operations like fgetc() and fprintf(). It is also recommended to handle larger chunks of samples, like 32, at a time.

Stereo samples are stored in an interleaved fashion. In 32-bit mode, the least significant word is stored first. This is the same as the native VSDSP 32-bit word order.

| **Audio sample buffer 16-bit word order** | | | | |
|---|---|---|---|---|
| **Audio format** | **Word 0** | **Word 1** | **Word 2** | **Word 3** |
| **16-bit stereo** | Left 0 | Right 0 | Left 1 | Right 1 |
| **32-bit stereo** | Left 0 LSW | Left 0 MSW | Right 0 LSW | Right 0 MSW |

## 6.2 VSOS Audio Output Example Program

The following audio program example creates a low-intensity sine wave to the left channel, then outputs the samples.

```
#include <vo_stdio.h>
#include <stdlib.h>
#include <math.h>
#include <saturate.h>
#include <apploader.h>

#define SIN_TAB_SIZE 96
#define SIN_AMPLITUDE 1000 /* Max 32767 */

static const s_int16 __y sinTab[SIN_TAB_SIZE];

int main(void) {
  // Remember to never allocate buffers from stack space. So, if you
  // allocate the space inside your function, never forget "static"!
  static s_int16 myBuf[2*SIN_TAB_SIZE];
  int i;

  /* Build sine table */
  for (i=0; i<SIN_TAB_SIZE; i++) {
    sinTab[i] = (s_int16)(sin(i*2.0*M_PI/SIN_TAB_SIZE)*SIN_AMPLITUDE);
  }

  while (1) {
    // Clear buffer
    memset(myBuf, 0, sizeof(myBuf));

    // Create sine wave to the left channel.
    for (i=0; i<SIN_TAB_SIZE; i++) {
      myBuf[i*2] = sinTab[i];
    }

    // Write result
    fwrite(myBuf, sizeof(s_int16), 2*SIN_TAB_SIZE, stdaudioout);
  }

  // Not really needed because there was a while(1) before
  return EXIT_SUCCESS;
}
```

### 6.3 VSOS Audio Input/Output Example Program

The following audio program reads audio from the default input, and sends it to the default output, until the user pushes Ctrl-C in the VSOS Shell Environment.

```
#include <vo_stdio.h>
#include <apploader.h> // Contains LoadLibrary() and DropLibrary()
#include <consolestate.h>

#define BUFSIZE 128

ioresult main(char *parameters) {
  static s_int16 myBuf[BUFSIZE];

  if (!stdaudioin || !stdaudioout) {
    printf("E: NO AUDIO IN OR OUT!\n");
    return S_ERROR;
  }

  while (!(appFlags & APP_FLAG_QUIT)) { /* Until Ctrl-C is pushed */
    fread(myBuf, sizeof(s_int16), BUFSIZE, stdaudioin);
    fwrite(myBuf, sizeof(s_int16), BUFSIZE, stdaudioout);
  }

  return S_OK;
}
```

# 7 Audio Drivers

VS1005g has multiple audio paths. This Chapter will explain which driver you will need to attach each audio driver to your software.

## 7.1 General

Audio drivers are named using the following format:
`AUdyyyyyz.DL3`
where

| Symbol | Description |
|---|---|
| d | Driver direction: I = input, O = output, X = Input+Output |
| yyyyy | Driver name, max. 5 characters |
| z | Optional M or S if e.g. I2S driver is Master or Slave |

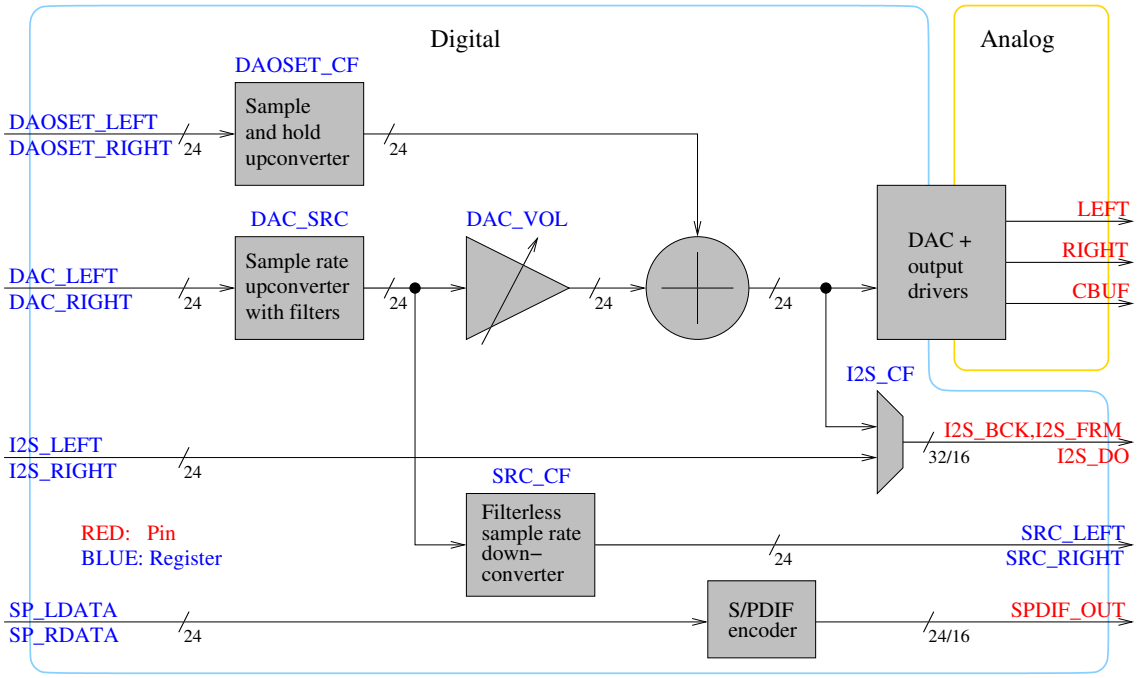


Figure 1: VS1005g playback (DA) audio paths

Figure 1 shows the VS1005 hardware output audio paths. Most of these have a driver controlling them.

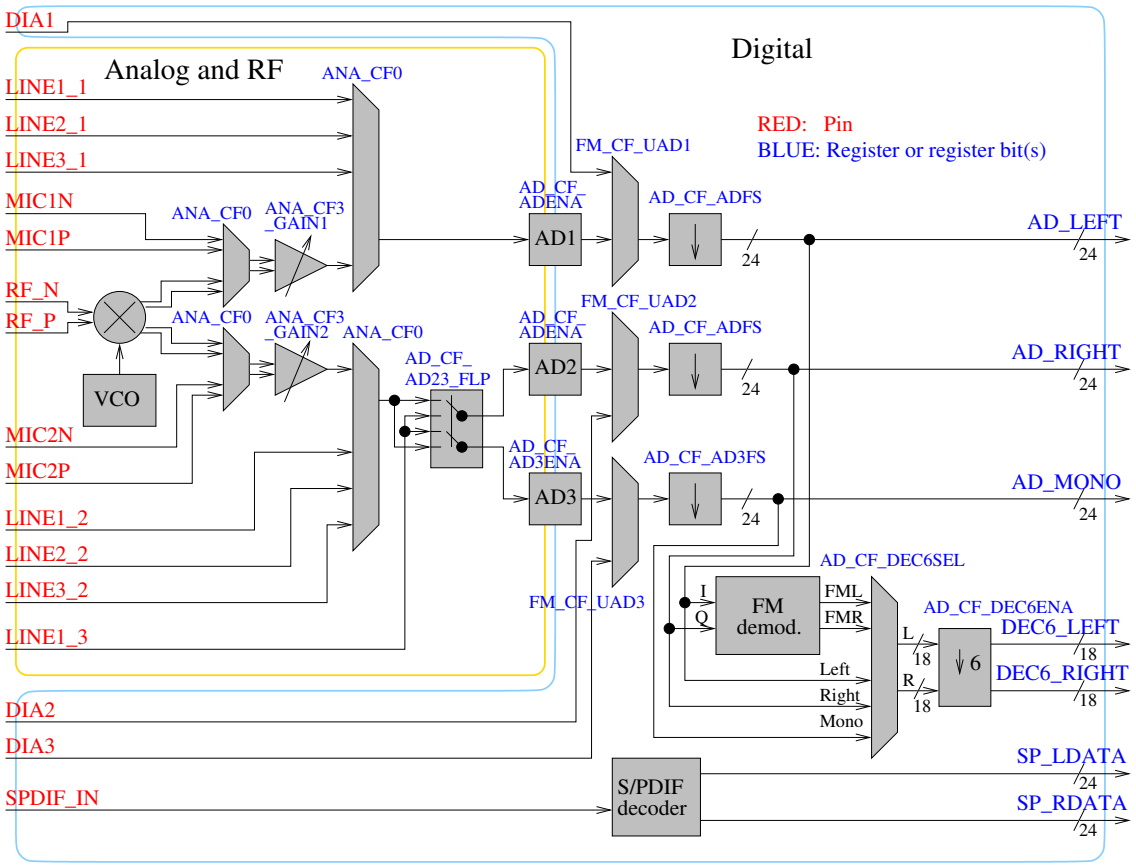


Figure 2: VS1005g recording (AD, FM, etc) signal paths

Figure 2 shows the VS1005 hardware input audio paths. Many of these have a driver controlling them.

## 7.2 Analog Output DAC Audio Drivers

### 7.2.1 Driver AUODAC.DL3

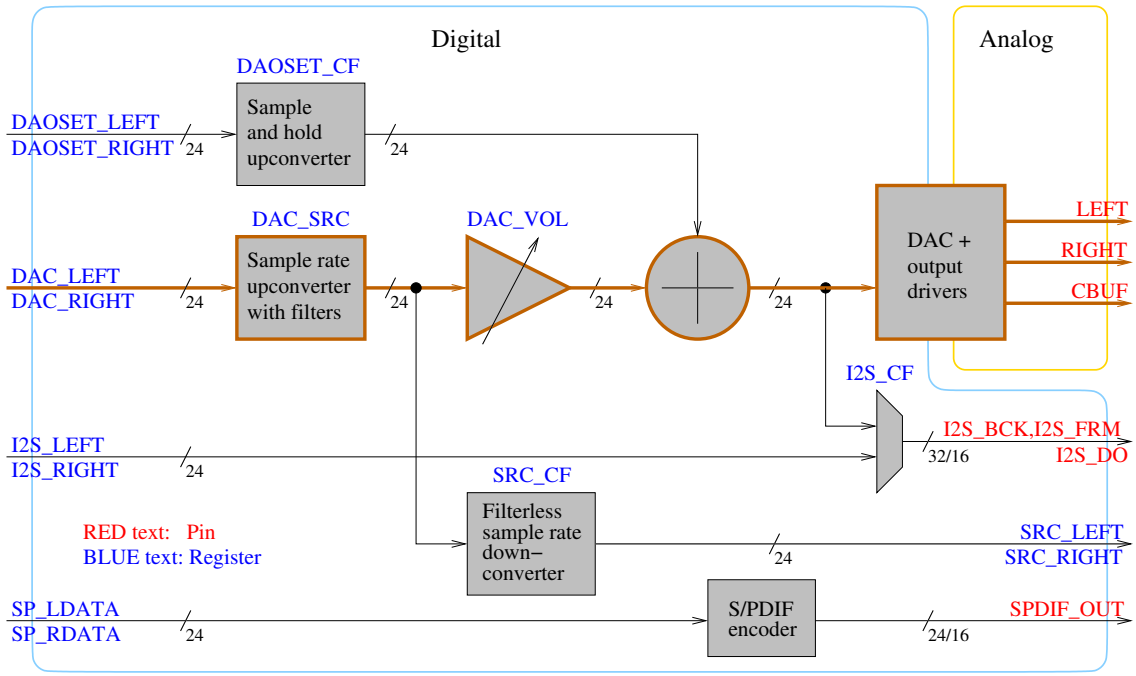


Figure 3: AUODAC.DL3 signal paths shown in **bold brown**

Figure 3 shows the VS1005 high-quality, fully filtered analog output main audio path.

AUODAC.DL3 is the basic DAC output driver. It takes over the VSOS default driver and offers a lot of funtionality over it, like 16-bit and 32-bit data transfers. It takes over *stdaudioout*, so all software that writes to standard output will send audio to this driver.

The driver offers setting the sample rate with an approximately 0.09 Hz accuracy between 100 and 97500 Hz on VS1005g. On VS1005h, sample rate accuracy is generally better. Audio is upconverted to an extremely high rate of 6.144 MHz by a high-quality hardware sample rate upconverter.

Playback volume can be set with 0.5 dB accuracy between full level volume (-0 dB) and -127 dB.

Note: On VS1005g, and using the standard 12.288 MHz crystal, some standard sample rates can be played back accurately (or as accurately as the system clock crystal runs), while some others have slight rounding errors. E.g. 96000, 48000, and 24000 Hz can be played back exactly. However, 44100 Hz is played back at ≈44100.0366 Hz, and 8000 Hz is played back at ≈7999.9695 Hz. While these errors are of the same order of magnitude as crystal speed errors and usually insignificant, intime they can break internal sync between the ADC and DAC converters. When run on VS1005h, the driver uses a “fractional sample rate” hardware feature that allows it to play such standard sample rates as 8000, 16000, 32000, 11025, 22050, and 44100 accurately.

## 7.3 Analog Side Path Audio Drivers

### 7.3.1 Driver AUOOSET.DL3

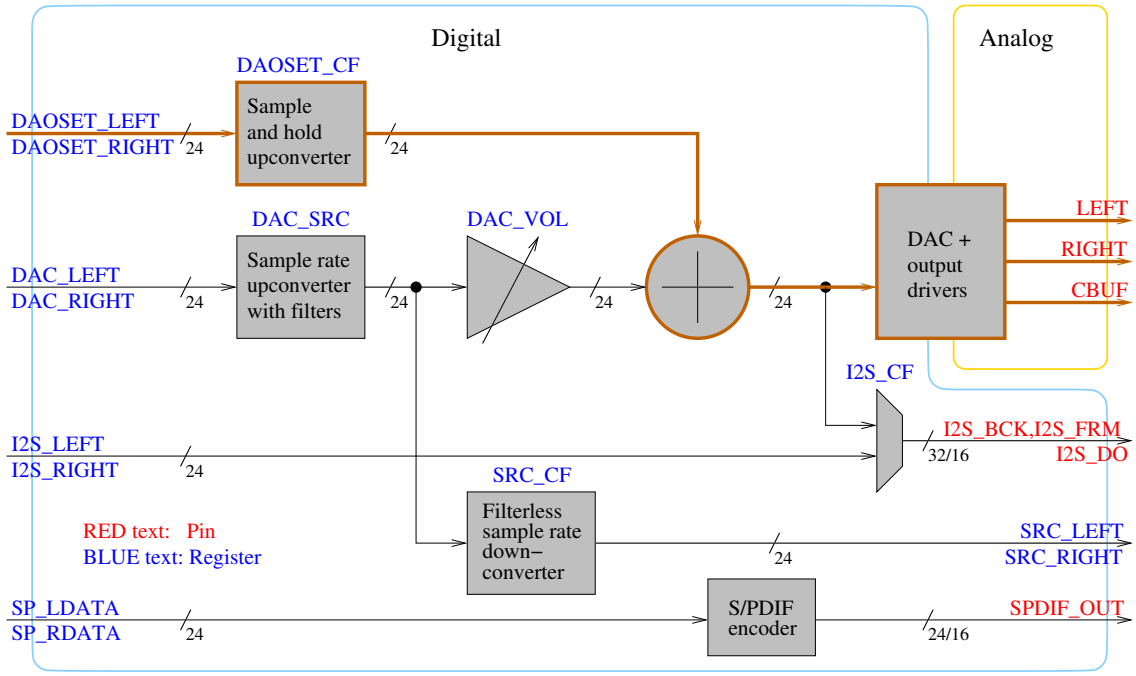


Figure 4: AUOOSET.DL3 signal paths shown in **bold brown**

Figure 4 shows the VS1005 analog output audio side path. This audio path is not filtered; it is only put through a Sample and hold upconverter. As such, audible aliasing distortion may be heard if low sample rates are used. This audio path is best suitable for different kinds of alarm and effects sounds that may easily be independently overlayed on top of the audio of the main audio path (see Chapter 7.2.1).

The sample rate of the side audio path is independent from the main audio path. While it may be set to up to 192 kHz, all sample rates cannot be set accurately. While certain sample rates like 24, 48, and 96 kHz can be played accurately, some others, like 44.1 kHz, may have an up to 150 Hz error. While not a problem for effects sounds, this may be create issue with accurate timing when playing longer audio passages.

While there is no hardware volume control for the side audio path, the driver offers an equivalent software volume control.

## 7.4 Analog Input ADC Audio Drivers

### 7.4.1 Driver AUIADC.DL3

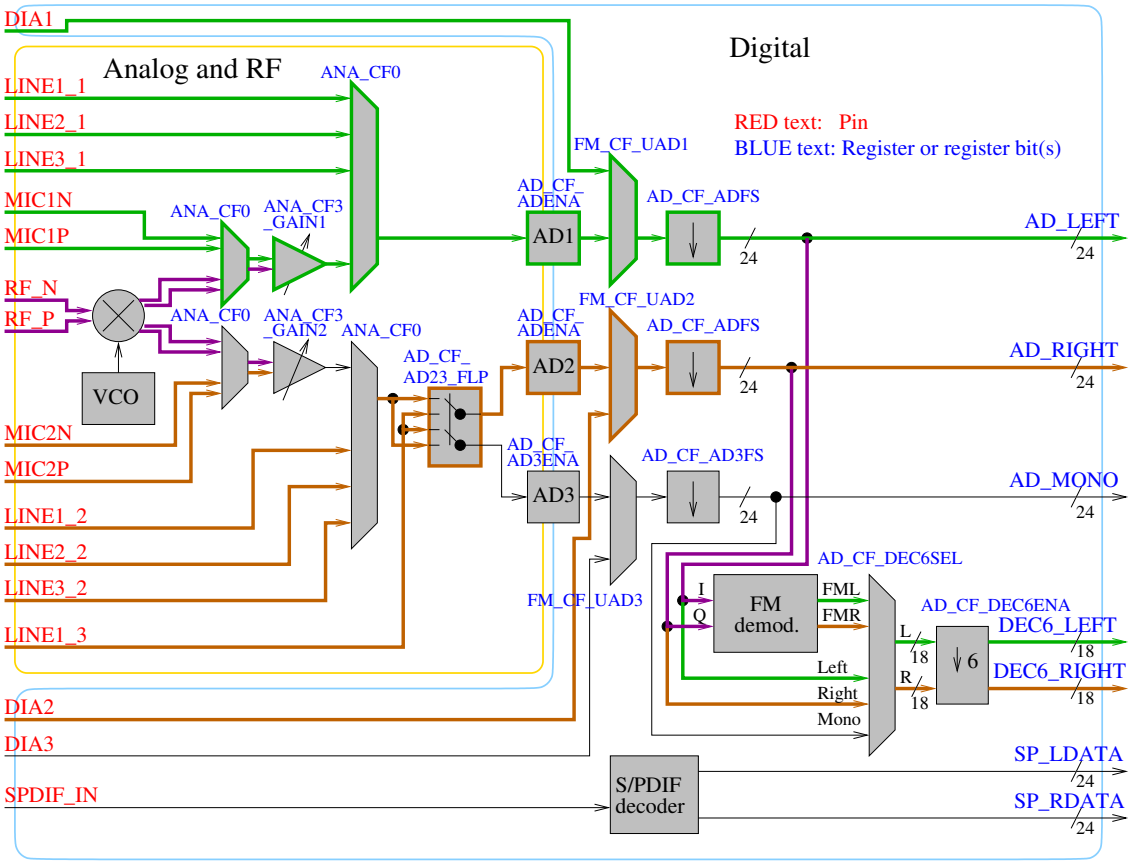


Figure 5: AUIADC.DL3 selectable input signal paths shown in **bold green** for the left channel, and **bold brown** for the right channel. Alternative RF audio path shown in **bold magenta**

The AUIADC.DL3 driver lets the user select a stereo input from a multitude of analog (and even some digital) sources. The sources used may be chosen at startup time, or changed dynamically while the driver is running. Any **brown** source in Figure 5 may be combined with any **green** source to form a stereo signal. However, if the **magenta-coloured** RF input is selected, it takes over the whole stereo audio path.

Supported sample rates are 192, 96, 48, and 24 kHz. However, it is also possible to use a high-quality down-by-6 decimator to create such sample rates as 32, 16, and 8 kHz. When the decimator is selected, the driver automatically reads its samples from the DEC6_LEFT/DEC6_RIGHT registers instead of the default AD_LEFT/AD_RIGHT.

Note that even if RF is selected for FM radio input, all of the FM hardware is not started by the driver. So you will still need a dedicated FM Receiver program to e.g. tune the FM radio. The only supported sample rate for the FM receiver is 32 kHz (using 192 kHz main sample rate and putting the signal through the I/Q - FM demodulation - down-by-6

decimator hardware).

Optionally, digital microphone inputs DIA1 and DIA2 may be used instead of the analog inputs. The 1-bit signals in the megahertz domain from the microphones are fed to the high-quality digital low-pass filtering path of VS1005.

On the VS1005 DevBoard, LINE1_1 and LINE1_3 are used as the default analog inputs. On the VS1005 BreakOut Board, LINE1_1 and LINE1_2 are used.

The input can be controlled using the VSOS Shell environment using the AUINPUT program (Chapter 9.1).

## 7.5 I2S Audio Drivers

I2S Audio drivers allows for I2S operation in both master and slave mode. Whenever possible, it is recommended to use master mode, because that way VS1005 has exact control over the sample rate.

The sample rate and number of bits (16/32) may be controlled with ioctl() commands IOCTL_AUDIO_SET_RATE_AND_BITS (recommended), IOCTL_AUDIO_SET_IRATE, IOCTL_AUDIO_SET_ORATE, and IOCTL_AUDIO_SET_BITS. In master mode, sample rates 24, 48, 96, and 192 kHz are supported.

In slave mode, the other end selects the sample rate, which is the same for both I2S input and output. If the user wants to monitor audio using analog output, they need to use the Slave Audio Input Synchronization Driver (Chapter 7.7).

With the exception of AUOI2SMA.DL3, all drivers connect to *stdaudioin* and/or *stdaudioout* if started with parameter “s”. Otherwise, the drivers need to be opened and accessed manually.

### 7.5.1 Driver AUOI2SMA.DL3

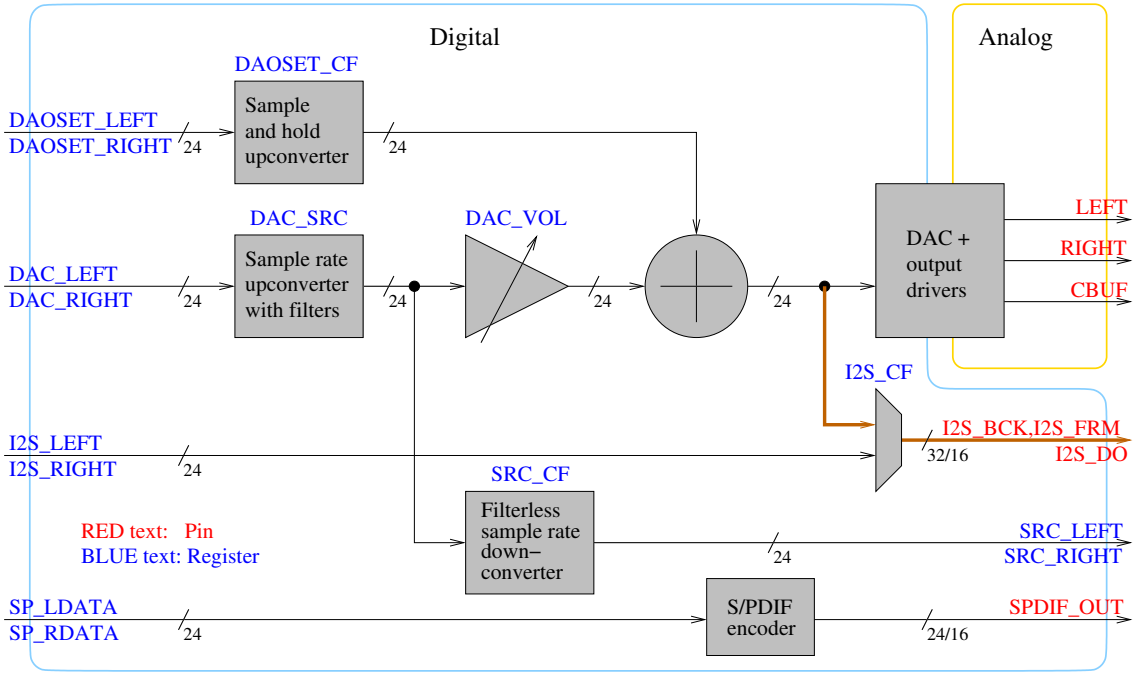


Figure 6: AUOI2SMA.DL3 audio path shown in **bold brown**

Figure 6 shows the automatic audio path activated by the driver. The driver copies the sum of the DAC and DAOSET drivers, with volume applied to the DAC contents, and sends them to I2S. To function, it needs a DAC (e.g. AUODAC.DL3) and/or DAOSET (e.g. AUOOSET.DL3) driver to be installed.

The sample rate is set to a default of 96000 Hz / 32 bits. Anything played back through VS1005's analog audio path is converted to the target sample rate by VS1005 hardware.

### 7.5.2 Driver AUOI2SM.DL3

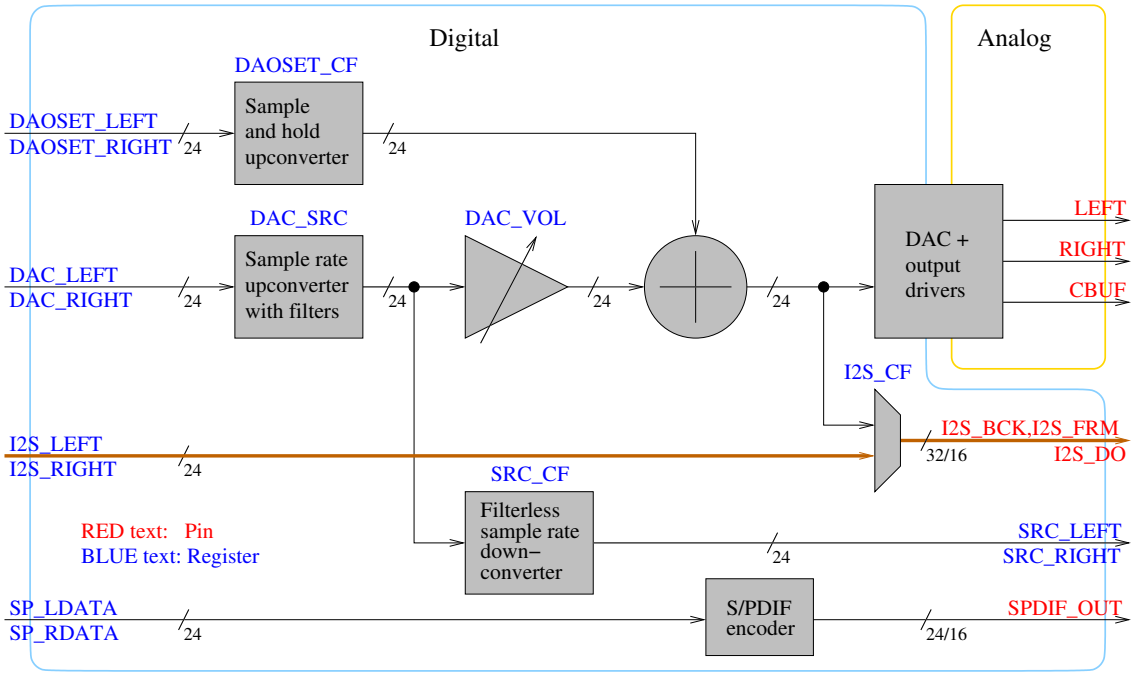


Figure 7: AUOI2SM.DL3 audio path shown in **bold brown**

If you need to send independent audio to the DAC and I2S, using AUOI2SM.DL3 is required. Note, however, that that driver can only support the basic master mode sample rates (e.g. not 44100 Hz).

Figure 7 shows the manual audio path activated by the AUOI2SM.DL3 driver.

### 7.5.3 Driver AUOI2SS.DL3

The AUOI2SS.DL3 is otherwise similar to AUOI2SM.DL3 (Chapter 7.5.2), except that the driver operates in slave mode.

In slave mode the user has no control over sample rate, so the audio cannot be fed anywhere else except the I2S output without resynchronization. Currently there does not exist a driver to synchronize I2S slave output with DAC output. Also there is no driver to synchronize inputs with I2S slave output.

### 7.5.4 Driver AUII2SM.DL3

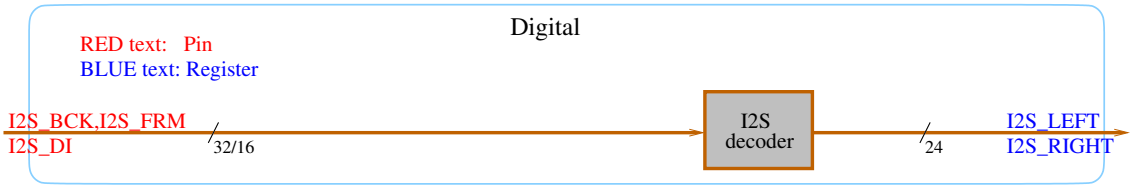


Figure 8: AUII2SM.DL3 audio path shown in **bold brown**

Figure 8 shows the audio path activated by the AUII2SM.DL3 driver, which is a master mode input driver.

### 7.5.5 Driver AUII2SS.DL3

The AUII2SS.DL3 is otherwise similar to AUII2SM.DL3 (Chapter 7.5.4), except that the driver operates in slave mode.

In slave mode the user has no control over sample rate, so the audio cannot be fed anywhere else except the I2S output without resynchronization. To synchronize I2S slave audio with the analog audio output driver AUODAC.DL3 (Chapter 7.2.1), use the AUXSYNCS.DL3 synchronization driver (Chapter 7.7.1).

### 7.5.6 Driver AUXI2SM.DL3

The AUXi2SM.DL3 audio driver handles both I2S input and output in master mode, as shown in Figures 7 and 8.

The I2S input and output are always kept in sync, so software using both the input and output doesn't need to do synchronization. Also, because the exact I2S sample rates 24 and 48 kHz are directly supported by VS1005's analog audio output path, as well as the analog audio input path, once in sync they will stay in sync.

### 7.5.7 Driver AUXI2SS.DL3

The AUXI2SS.DL3 is otherwise similar to AUXI2SM.DL3 (Chapter 7.5.6), except that the driver operates in slave mode.

In slave mode the user has no control over sample rate, so the audio cannot be fed anywhere else in realtime, except the I2S output without resynchronization. To synchronize I2S slave audio with the analog audio output driver AUODAC.DL3 (Chapter 7.2.1), use the AUXSYNCS.DL3 synchronization driver (Chapter 7.7.1).

## 7.6 S/PDIF Audio Drivers

S/PDIF drivers allow receiving and transmitting audio using a coaxial or optical S/PDIF bus.

S/PDIF is a one-way channels, so the transmitter always controls the clocking of the audio. For this reason, and for certain hardware resons, those drivers that need to either support synchronizing S/PDIF output to input, or to support 44.1 ot 88.2 kHz sample rates, need to set VS1005's master clock to either 56.448 or 61.440 MHz.

With the exception of AUOXSPDA.DL3, all drivers connect to *stdaudioin* and/or *stdaudioout* if started with parameter "s". Otherwise, the drivers need to be opened and accessed manually.

Only one S/PDIF driver may be in use at any given time. It is not possible to run one input and one output driver, and expect the system to do anything sensible.

All S/PDIF drivers are incompatible with USB operation, because USB requires that CLKI = 60.000 MHz, and no standard sample rates can be implemented with this clock rate.

| Summary of S/PDIF drivers | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Name** | **Sample rates / kHz** | | | | **Modifies SysClk**[1] | **Exact rates**[2] | **Notes** |
| | **44.1** | **48** | **88.2** | **96** | | | |
| AUOSPDA | | X | | X | | X | [3] Automatic output |
| AUISPD | X | X | X | X | X | | Generic input |
| AUXSPD | X | X | X | X | X | | Generic I/O, sync in & out |
| AUXSPD48 | | X | | X | X | | Limited I/O, sync in & out |
| AUOSP48S | | X | | | X | | Output, sync with input |
| AUOSPD48 | | X | | | | X | Output at exactly 48 kHz |

[1] The driver modifies System Clock either 56.448 MHz (for 44.1 or 88.2 kHz operation) or 61.440 MHz (for 48 or 96 kHz operation). The driver is incompatible with FM receiver software.

[2] The driver sample rate is as accurate as the input clock's 12.288 MHz is.

[3] Audio sample rate may be anything up to 96 kHz. It is converted by hardware to the target sample rate.

### 7.6.1 Driver AUOSPDA.DL3

AUOSPDA.DL3 captures audio that is being sent to the DAC audio path, and copies it automatically to the S/PDIF output.

Figure 9 shows the automatic audio path built by the driver. The driver copies the audio going to the DAC driver, and sends it through a software volume control to the S/PDIF output. To function, it needs a DAC (e.g. AUODAC.DL3) driver to be installed.

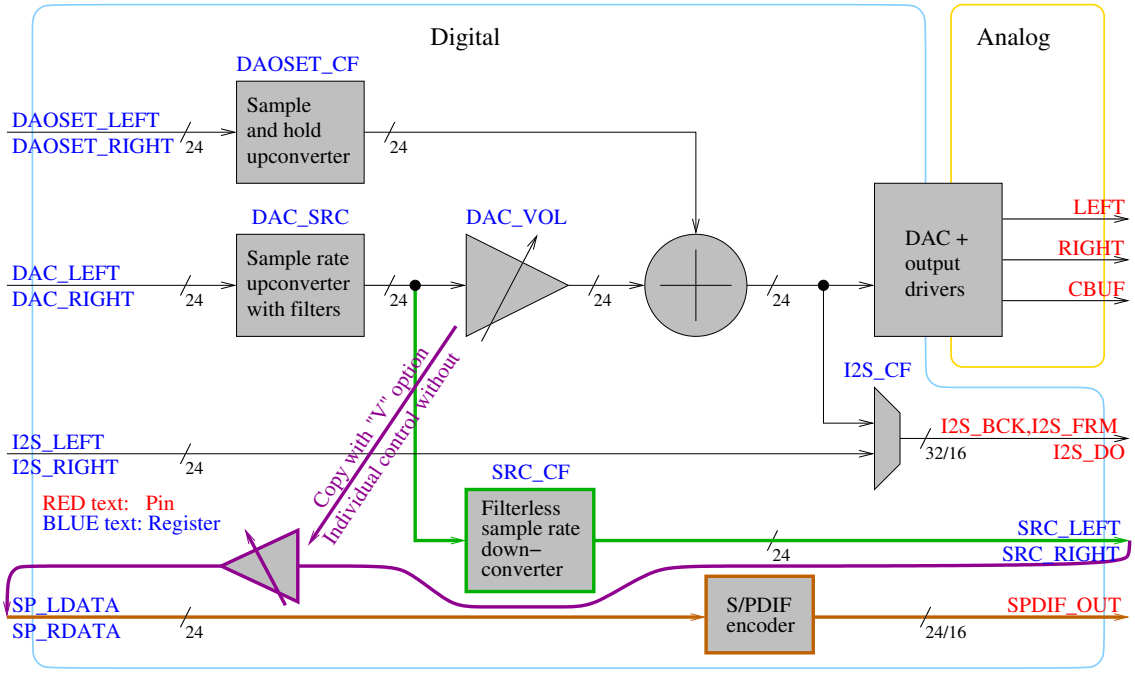


Figure 9: AUOSPDA.DL3 audio path shown in **bold brown**. Software driver connecting to the filterless sample rate converter (**bold green**) shown in **bold magenta**

The sample rate is set by default to 96000 Hz, and anything played back is converted to the target sample rate by the high-quality VS1005 Filterless sample rate downconverter, then through a software volume control to the SP_LDATA/SP_RDATA registers. As long as the audio that is being played back has a sample rate that is not higher than the S/PDIF output sample rate, no aliasing will occur, and sound quality will remain good.

If the driver is started with parameter “48000”, it will use an output rate of 48 kHz. Only use this option if no audio that you play back has a sample rate over 48 kHz.

The driver can be opened manually, in which case it has a separate volume control that can be used. As a default, full output volume is used. However, if the driver is started with parameter “v”, it will automatically copy any volume setting sent to *stdaudioout*.

If the driver has not started with the “v” option, the user may set the volume from C by opening a file pointer to it, then calling stdio() to set the volume (see README.TXT of the driver for details on how to do that), then finally closing the file pointer / driver. Alternatively volume may be set from the VSOS Shell using AuOutput (Chapter 9.2), as in the following example that sets volume to -12.5 dB of maximum level:

```
S:>auoutput -dauospda -l-12.5
```

### 7.6.2 Driver AUISPD.DL3

AUISPD.DL3 enables S/PDIF input. It supports 44.1 kHz, 48 kHz, 88.2 kHz, and 96 kHz sample rates.

This driver automatically sets the VSDSP core clock to enable S/PDIF operation. It is incompatible with any USB drivers or FM Radio receiver software.

Example of how to activate under VSOS Shell so that input from S/PDIF is automatically played back to the DAC, and the DAC is kept synchronized with the input:

```
S:>driver +auodac s
S:>driver +auispd s
S:>driver +auxsyncs
S:>driver +auxplay
S:>auinput
stdaudioin:      0x23dc, auispd::audioFile=0x0c63(3171)
  ->Identify():  0x43b7, auxsyncs::Identify returns "AUXSYNCS"
  ->op:          0x23e3, auispd::audioFileOps=0x0000(0)
    ->Ioctl():   0x41ee, auxsyncs::AudioIoctl
    ->Read():    0x40f6, auispd::AudioRead
Sample rate:     47995
Bits per sample: 16
Buffer size:     512 16-bit words (256 16-bit stereo samples)
Buffer fill:     84 16-bit words (42 16-bit stereo samples)
Sample counter:  1068008
Overflows:       4170
S:>
```

### 7.6.3 Driver AUXSPD.DL3

AUXSPD.DL3 enables S/PDIF input and output, and synchronizes the output with the input. It supports 44.1 kHz, 48 kHz, 88.2 kHz, and 96 kHz sample rates.

Because this driver automatically synchronizes its output with its input, there is no need to run a separate synchronization driver like AUXSYNCS.DL3.

This driver automatically sets the VSDSP core clock to enable S/PDIF operation. It is incompatible with any USB drivers or FM Radio receiver software.

Example of how to activate under VSOS Shell so that input from S/PDIF is automatically played back to the S/PDIF output:

```
S:>driver +auxspd s
S:>driver +auxplay
S:>auinput
stdaudioin:      0x279f, auxspd::audioFile=0x0c63(3171)
  ->Identify():  0x40ed, auxspd::Identify returns "AUXSPD"
  ->op:          0x27a7, auxspd::audioFileOps=0x0000(0)
    ->Ioctl():   0x3f69, auxspd::AudioIoctl
    ->Read():    0x4059, auxspd::AudioRead
Sample rate:     48010
Bits per sample: 16
Buffer size:     512 16-bit words (256 16-bit stereo samples)
Buffer fill:     82 16-bit words (41 16-bit stereo samples)
```

```
Sample counter:  654782
Overflows:       1320
S:>
```

### 7.6.4 Driver AUXSPD48.DL3

Like AUXSPD.DL3 (Chapter 7.6.3), but limited to 48 and 96 kHz, and for this reason slightly smaller.

### 7.6.5 Driver AUOSP48S.DL3

AUOSPD48S.DL3 enables S/PDIF output, and synchronizes the output with an input of another audio driver, which should nominally run at 48 kHz. An example use case for this driver is when the input is I2S in slave mode (Chapter 7.5.5, *Driver AUII2SS.DL3*), and when it is known beforehand that the nominal sample rate of the input is always 48 kHz.

Because this driver automatically synchronize its output with an input, there is no need to run a separate synchronization driver like AUXSYNCS.DL3.

This driver automatically sets the VSDSP core clock to enable S/PDIF operation. It is incompatible with any USB drivers or FM Radio receiver software.

Example of how to activate under VSOS Shell so that input from I2S (slave mode) is automatically played back to the S/PDIF output (note: nothing is played through analog audio): the input:

```
S:>driver +auii2ss s
S:>driver +auosp48s s
S:>driver +auxplay
S:>
```

### 7.6.6 Driver AUOSPD48.DL3

AUOSPD48.DL3 enables S/PDIF output at exactly 48 kHz.

Example of how to activate under VSOS Shell so that input from the ADC is automatically played back to the S/PDIF output (note: nothing is played through analog audio):

```
S:>driver +auiadc s
S:>driver +auospd48 s
S:>driver +auxplay
S:>
```

## 7.7 Slave Audio Input Synchronization Drivers

When inputting audio data in slave mode (using for example the I2S audio input slave driver AUII2SS.DL3), the exact sample rate of the audio is usually not known. Even if the nominal sample rate is known, mismatches between master transmitter and the VS1005 receiver clock crystals causes there to always be a mismatch between them (example: transmitter nominally sends 48000 Hz, but because of a clock mismatch the receiver sees the data at 48002.3 Hz).

This speed mismatch will eventually cause an audio buffer underflow or overflow, which may cause audible clicks or other kinds of distortion.

The slave audio input synchronization drivers are intended to remove the synchronization issue.

### 7.7.1 Driver AUXSYNCS.DL3

The Slave Audio Input Synchronization Driver AUXSYNCS.DL3 synchronizes a slave audio input driver with the analog Earphone/Line Out driver AUODAC.DL3.

Before starting the Sync Driver, the user must first load and connect a slave audio input driver to *stdaudioin*, and the analog output driver to *stdaudioout*. When the driver is loaded, it will automatically adjust the analog output sample rate according to the input. The adjustment range is up to 97500 Hz, so standard sample rates up to 96 kHz can be received. The Sync Driver can dynamically change its sample rate if the input sample rate changes.

Example config.txt file clip:

```
# Load I2S Slave Input driver and make it stdaudioin
AUII2SS s
# Load Line Out / Earphone output driver and make it stdaudioout
AUODAC s
# Connect and synchronize stdaudioout with stdaudioin slave
AUXSYNCS
```

The same can be done using the VSOS Shell using the following commands:

```
S:>driver +auii2ss s
S:>driver +auodac s
S:>driver +auxsyncs
```

AUXSYNCS.DL3 has been tested with the I2S Slave Input drivers, but it is designed to be usable with any generic slave input driver that offers a near-constant data rate. It may not work properly with input drivers with large data bursts.

## 7.8 Audio Input to Output Copying Driver

Sometimes it's useful to play back audio data from an input to an output in the background. This can be done by an audio copying driver.

### 7.8.1 Driver AUXPLAY.DL3

The AUXPLAY.DL3 driver reads data from *stdaudioin* and copies it to *stdaudioout*. While seemingly trivial, it does so in the background, allowing the user to do other operations while sound is being played back.

Normally the driver reports to *stdout* if there are input buffer overflows or output buffer underflows. The amount of the overflows/underflows are given in stereo samples (so e.g. +4800 at a sample rate of 48000 means 1/10s). The reports use the following format:

```
AUXSPLAY: In overflow +4088
AUXSPLAY: Out underflow +4034
```

To disable overflow and underflow reporting, give the 'q' parameter to AUXSPLAY.DL3.

# 8 Audio Filter Drivers

Audio filter drivers connect to an audio source or sink, and offer additional functionality, like filtering.

Audio filter drivers are named using the following format:
`FTdyyyyy.DL3`
where

| Symbol | Description |
|---|---|
| d | Driver direction: I = input, O = output, X = Input/Output |
| yyyyy | Driver name, max 5 characters |

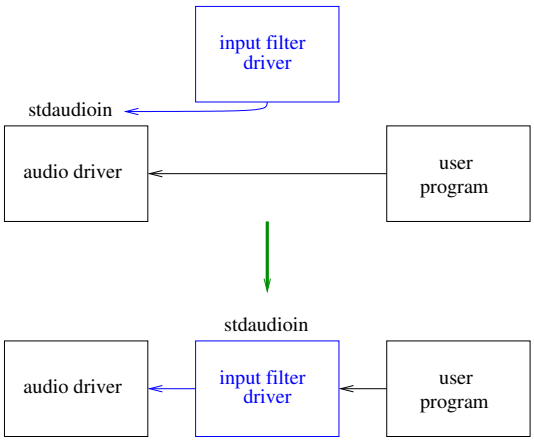


Figure 10: A filter input driver connects to the stdaudioin chain

All filter input drivers connect directly between the current *stdaudioin* program chain and the user program, as shown in Figure 10. It is important to first start the base driver responsible for *stdaudioin* before starting the filter driver!

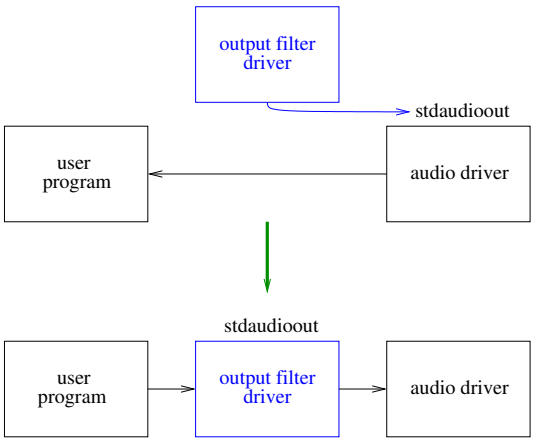


Figure 11: A filter output driver connects to the *stdaudioout* chain

All filter output drivers connect directly between the user program and the current *stdaudioout* program chain, as shown in Figure 11. It is important to first start the base driver

responsible for *stdaudioout* before starting the filter driver!

If the user wishes to remove an audio driver or some filters, they always have to removed in reverse order as they were allocated. E.g. if AUODAC.DL3 and FTOEQU.DL3 have been loaded, they must be released in order FTOEQU.DL3, AUODAC.DL3.

## 8.1 Equalizer Audio Drivers

The Equalizer audio drivers allow for multiband equalization to be implemented to the VS1005's output audio path.

The package itself contains detailed PDF documentation; please read that for details.

### 8.1.1 Driver FTOEQU.DL3

FTOEQU.DL3 connects the equalizer driver to *stdaudioout*. Read the PDF documentation for FTEQU for more details.

### 8.1.2 Control Program SETEQU.DL3

```
Usage: SetEqu [-i|-o] [n [flags centerF gain qFactor]] [-h]
-i      Set stdaudioin
-o      Set stdaudioout (default)
n       Use filter number n (1 ... MAX_FILTERS)
flags   1=left, 2=right
centerF Center frequency in Hz
gain    Gain in dB (-12.0 ... 12.0)
qFactor Q Factor (0.1 ... 4.0)
-h      Show this help
```

Examples:

```
setequ 1 3 400 -6.0 0.5 # Set filter 1, L+R, 400 Hz, -6 dB, Q 0.5
setequ 1 0              # Clear filter 1
setequ 1                # Show filter 1
setequ                  # Show all filters
```

SETEQU.DL3 is a program to set and/or display the equalizer parameters.

Note that the equalizer is a relatively expensive function, so more than a bass/treble controller should only be used with care.

The full documentation for the software is in the PDF file for the FtEqu package.

## 8.2 DC Offset/AGC Audio Drivers

When audio is digitized, two technical issues are DC Offset and Large Dynamic Range.

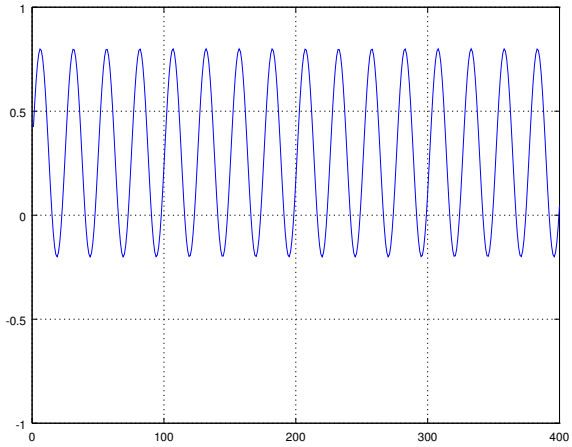


Figure 12: Audio with exaggerated DC offset

In an ideal world DC Offset wouldn't happen. However, in the real world, signals almost always have a slight DC offset. Note, how the sine wave in Figure 12 does not move evenly around the center point, but has an offset of about +0.35. While the figure has been greatly exaggerated, this is a real phenomenon caused by a myriad of different reasons.

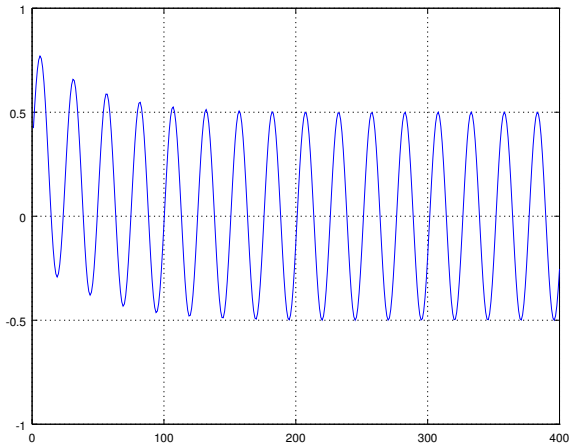


Figure 13: Audio with DC blocking

DC offset may cause many issues, including increased power consumption, audible cracks and pops, waring down of speaker elements, and non-ideal audio compression. Because of this, it is best to remove the audio offset with a DC Blocker algorithm, as shown in Figure 13. Notice how the offset disappear after a little while (in this case, it

has vanished practically completely by sample 150).

Another issue in audio is excessive dynamic range. This is not a problem when recording well-mixed, pre-recorded music, but it may be a big issue when recording speech from the microphone. To compensate for the audio level differences of close and faw away speakers, and Automatic Gain Control (AGC) unit may often be useful. Note, however, that AGC should not be used for HiFi recording applications!

### 8.2.1 Driver FTIDCBL.DL3

The DC Block driver FTIDCBL.DL3 connects to *stdaudioin* as shown in Figure 10 on page 28.

The driver may be controlled either through C ioctl() function calls as described in the README.TXT file for the driver itself, or from the VSOS Shell using the SETAGC.DL3 command.

### 8.2.2 Driver FTIAGC.DL3

In addition to the DC Block driver described in 8.2.1, the AGC driver offers an Automatic Gain Control function.

The driver may be controlled either through C ioctl() function calls as described in the README.TXT file for the driver itself, or from the VSOS Shell using the SETAGC.DL3 command.

### 8.2.3 Control Program SETAGC.DL3

```
Usage: SetAgc [-i|-o] [-a x|-d x|-t x|-max x|-min x|-d x] [-h]
-i      Set stdaudioin (default)
-o      Set stdaudioout
-a x    Set attack (ms)
-d x    Set decay (ms)
-t x    Set target level (dB)
-max x  Set maximum gain (dB)
-min x  Set minimum gain (dB)
-b x    Set DC Block Filter (0x4000 = HiFi, 0x8000 = Speech,
                             0x0    = Auto, 0xC00x = Set to x)
-h      Show this help

With no parameters SetAgc will show current values
```

Sets/Prints AGC and/or DC Block Filter parameters. For the DC Block Filters only the -b setting option is available.

## 8.3 Pitch Shifter / Speed Shifter Audio Drivers

The FtPitch package offers a pitch and speed shifter that connects to stdout. The speed shifter can nominally be controlled to speeds between 0.68x and 1.64x of realtime, and the pitch shifter can nominally be controlled between 0.61x and 1.47x of normal pitch.

Features and limitations:

- Speed divided by pitch (speed/pitch) must be between 0.68 and 1.64.
- Pitch shifting alters sample rate. If the resulting sample exceeds 96 kHz, playback will be at incorrect speed.
- The shifter has been optimized to work best for audio where sample rate is between 32 and 48 kHz.

### 8.3.1 Driver FTOPITCH

The Pitch shifter driver FTOPITCH.DL3 connects to *stdaudioout* as shown in Figure 11 on page 28.

The driver may be controlled either through C ioctl() function calls as described in the README.TXT file for the driver itself, or from the VSOS Shell using the SETPITCH.DL3 command.

### 8.3.2 Control Program SETPITCH

```
Usage: SetPitch [-i|-o] [-sx] [-px] [-h]
-i      Set stdaudioin
-o      Set stdaudioout (default)
-sx     Set speed to x times normal (0.68 - 1.64 if pitch=1.0)
-px     Set pitch to x times normal (0.61 - 1.47 if speed=1.0)
-h      Show this help

Note:
Correct playback requires that 0.68 <= speed/pitch <= 1.644.
```

Sets/Prints Pitch and Speed Shifter parameters. Example:

```
S:>driver +ftopitch
S:>setpitch
Pitch 1.000
Speed 1.000
S:>setpitch -p1.1 -s0.9
S:>setpitch
Pitch 1.100
Speed 0.900
```

## 8.4 Reverb Generator Audio Drivers

The FtRev package offers a reverb-style echo generator that connects to stdout. Many parameters of the Reverb Generator may be modified.

### 8.4.1 Driver FTOREV

Features and limitations:

- Works up to 48 kHz 32 bits.
- Requires about 32 MIPS when running at 48 kHz 16 bits.

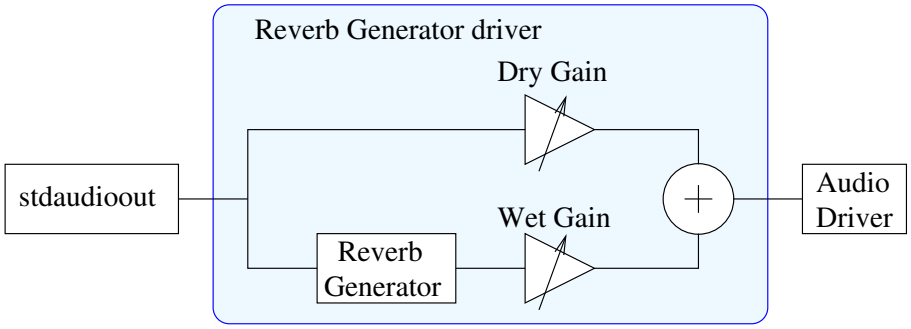


Figure 14: Reverb Generator FTOREV signal paths

The Reverb Generator driver FTOREV.DL3 connects to *stdaudioout* as shown in Figure 11 on page 28. The Reverb Generator signal paths are shown in Figure 14.

The driver may be controlled either through C ioctl() function calls as described in the README.TXT file for the driver itself, or from the VSOS Shell using the SETREV.DL3 command.

Note: To easily test how modifying parameters affects sound, you can start the automatic playback driver AUXPLAY (Chapter 7.8.1) as follows:

```
S:>driver +ftorev
S:>driver +auxplay
S:>setrev -v -s600 -d0 -w1024 -t65535 -f25000 -r85
ROOM:
  (-r) First reflection:     85 ms
  (-s) Room size       :    600 cm (1-32767)
  (-t) Reverb time     :  65535 ms (1-65535)
  (-f) Room softness   :  25000    (0=hard-65535=soft)
  (-d) Dry gain        :      0    (0-32767, 1024=1)
  (-w) Wet gain        :   1024    (0-32767, 1024=1)
       Sample rate     :  48000 Hz
```

```
        Delay pairs     :       7
        Ext. mem. size  :       0 bytes
        Ext. mem. read  : 0x0000
        Ext. mem. write : 0x0000
S:>
```

### 8.4.2 Driver FTOREV23

Features and limitations:

- FTOREV23 is like FTOREV (Chapter 8.4.1), but makes it possible to create larger rooms.
- Uses VS23S010 or VS23S040 S-RAM IC for buffering.
- Optimized for 48 kHz, but works up to 96 kHz mode.
- Requires about 48 MIPS when running at 48 kHz 16 bits.
- Requires about 59 MIPS when running at 96 kHz 32 bits.
- If sample rate is greater than 48 kHz (typ. 96 kHz), the first reflection option (-f in SETREV, Chapter 8.4.3) is not available.

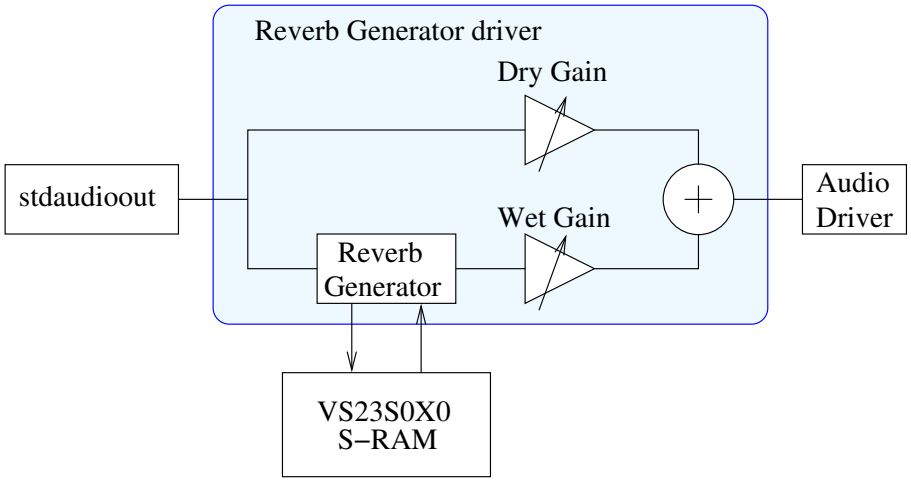


Figure 15: Reverb Generator FTOREV23 signal paths

The Reverb Generator driver FTOREV23.DL3 is like FTOREV.DL3 except that it uses an external VS23S010 or VS23S040 S-RAM IC for buffering, making it possible to create larger rooms than with FTOREV.DL3. The Reverb Generator signal paths are shown in Figure 15.

#### Running FTOREV23 at 96 kHz 32 bits

The FTOREV23 Reverb Generator supports sample rates up to 96 kHz and bit depths up to 32 bits. However, if used at these parameters, it requires lots of calculating power,

and when system overhead (typically 20 MIPS) is counted in, the clock requirement is typically around 80 MHz.

An example of how to start the driver in 96 kHz 32-bit mode succesfully from config.txt is shown below:

```
RUN SETCLOCK -193 80
AUODAC s
AUIADC s
RUN AUOUTPUT -b32 -r96000 -s2048
RUN AUINPUT -b32 -r96000 -s2048
FTOREV23
AUXPLAY
```

The same can be achieved from the command line as follows. Here you can also see that the system requires almost 79 MHZ to run:

```
S:>setclock -193 80
S:>driver +auodac s
S:>driver +auiadc s
S:>auoutput -b32 -r96000 -s2048
S:>auinput -b32 -r96000 -s2048
S:>driver +ftorev23
S:>driver +auxplay
S:>setclock -t
CPU speed: effective speed 7.33/86.02 MHz = 8.5%, overhead 78.68 MHz
S:>
```

### 8.4.3 Control Program SETREV

```
Usage: SetRev [-i|-o] [-v|+v] [-sx|-tx|-fx|-dx|-wx] [-h]
-i      Set stdaudioin
-o      Set stdaudioout (default)
-rx     Set first reflection time in milliseconds
-sx     Set room size to x cm (200-1200 recommended)
-tx     Set reverb Time to x ms (100-5000 recommended)
-fx     Set room wall soFtness (0 = hard, 65535 = soft)
-dx     Set Dry gain (0-65535, 1024 = 1)
-wx     Set Wet gain (0-65535, 1024 = 1)
-v|+v   Verbose on/off
-h      Show this help

Note:
It is recommended that Dry gain + Wet gain would not be much
over 1024 to avoid distortion.
```

```
S:>setrev
ROOM:
  (-r) First reflection:     20 ms
  (-s) Room size        :    600 cm (1-32767)
  (-t) Reverb time      :   1000 ms (1-65535)
  (-f) Room softness    :  32767    (0=hard-65535=soft)
  (-d) Dry gain         :   1024    (0-32767, 1024=1)
  (-w) Wet gain         :    256    (0-32767, 1024=1)
       Sample rate      :  48000 Hz
       Delay pairs      :      7
```

The user-adjustable parameters are as follows.

**First Reflection Time (-r, 1...200)**

First Reflection Time tells how many milliseconds it takes before the first reflection reaches the listener. By making this number large, the user can simulate a stadium-like environment or a hall where lots of reverberation comes from the other end of the hall.

**Room Size (-s, 1...32767)**

Room Size is the approximate room size in centimeters. The larger the room, the more indistinct, or hall-like, the reverb is.

Note that room size is limited by the amount of free data memory there is available in the system. When you run SetRev, the displayed parameter “Delay pairs” tells how well the reverb driver has been able to set its filters. In an ideal case, “Delay pairs” should be 8. Anything below 4 usually gives a somewhat artificial result. Also, if you define a room that is way too large to being implemented, it will automatically be resized.

In the example below, we try to make a “room” 300 meters wide and the first echo should come only after 500 milliseconds. But, depending on the driver we may end up with a room that is smaller, and with a shorter pre-echo time. Note also the “Delay pair” counter: it prefably be 8, and anything below 4 usually isn’t enough to create a natural result. Note that the driver FTOREV23.DL3 doesn’t have the room size limitation.

```
S:>setrev -s30000 -r500 -v
ROOM:
  (-r) First reflection:     85 ms
  (-s) Room size        :   9842 cm (1-32767)
  (-t) Reverb time      :   1000 ms (1-65535)
  (-f) Room softness    : 32767    (0=hard-65535=soft)
  (-d) Dry gain         :   1024    (0-32767, 1024=1)
  (-w) Wet gain         :    256    (0-32767, 1024=1)
```

```
Sample rate     : 48000 Hz
Delay pairs     :     1
```

**Reverb Time (-t, 1...65535)**

Reverb Time tells how many milliseconds it will take for the reverberation to fade -60 dB from the original level. For some insane effects, try to set the reverb time to 10 or 60 seconds!

**Room Softness (-f, 0...65535)**

Room Softness defines the softness of the wall material in the room. In a soft room, higher frequencies will attenuate faster than low frequencies, making the room feel warmer.

**Dry Gain (-d, 0...32768)**

Dry Gain is the gain setting for the direct input to output audio path. 1024 is the nominal gain: the original sound is passed through with no volume change.

To avoid distortion, it is recommended that Dry Gain + Wet Gain <= 1024, or at least not much over 1024.

**Wet Gain (-w, 0...32768))**

Wet Gain is the gain setting for the reverb signal. 1024 is the nominal gain.

To avoid distortion, it is recommended that Dry Gain + Wet Gain <= 1024, or at least not much over 1024.

## 8.5 Noise Killer Audio Drivers

Because of the way stereo information is transmitted on FM radio, stereo reception is always more suscept to white noise and other artifacts than mono reception. A way to reduce or remove the noise is to either dampen the stereo effect at the receiver, or to just turn FM stereo reception off. The FtNoiseKiller package offers an adaptive FM stereo radio noise killer algorithm that doesn't destroy the stereo image.

### 8.5.1 Driver FTINOISE

Features and limitations:

- Optimized for 32 kHz operation.
- Requires about 25 MIPS at 32 kHz 16 bits.
- Can only handle 16-bit audio (if audio is set to 32 bits, the noise killer is disabled).

The noise killer driver FTINOISE.DL3 connects to *stdaudioin* as shown in Figure 10 on page 28.

The driver may be controlled either through C ioctl() function calls as described in the README.TXT file for the driver itself, or from the VSOS Shell using the SETNOISE.DL3 command.

Note: To easily test how the algorithm works, you can start the RDS Radio receiver (v1.05 or higher) after activating the noise killer driver:

```
S:>driver +ftinoise
S:>rdsradio
```

Now you can turn the noise killer off by pushing '0', and back on by pushing '5' (or '4' or '3' if reception is poor).

### 8.5.2 Control Program SETNOISE

```
Usage: SetNoise [-i|-o] [-v|+v] [-nx] [-h]
-i      Set stdaudioin (default)
-o      Set stdaudioout
-nx     Set noise killer level (default: 50 dB, 0 = off)
-v|+v   Verbose on/off
-h      Show this help

S:>setnoise
```

```
Noise killer at 50 dB
S:>setnoise -n0 -v
Noise killer at 0 dB
S:>
```

The default for the noise killer is a signal-to-noise ratio of 50 dB. If the reception is poor, lower numbers may be required. Note that by the value of 30 dB, the stereo image mostly disappears. Setting the noise killer to 0 dB will disable it.

# 9 Audio Control Programs

These programs are useful for displaying and changing audio parameters, as well as debugging audio interfaces. They are parts of the AuControl solution.

## 9.1 Control Program AUINPUT.DL3

```
Usage: AuInput [-ddrv|-pfp|-rrate|-bbits|-sbufsize|chconf|-v|+v|-h]
-ddrv   Connect to audio driver DRV.DL3
-pfp    Set output audio driver pointer to fp (use with caution!)
-rrate  Set sample rate to rate
-bbits  Number of bits (16 or 32)
-sbufSz Set buffer size to bufSz 16-bit words
-v|+v   Verbose on|off
chconf  Audio channel config (only with AUIADC driver, see definitions below)
-h      Show this help

chconf needs either one stereo element, or one left and one right element.
Stereo elements:
- fm
Left elements:
- line1_1, line2_1, line3_1, mic1, dia1
Right elements:
- line1_2, line2_2, line3_2, line1_3, mic2, dia2, dia3

Example:
auinput line1_1 line1_3
```

AUINPUT lets the user display control several parameters of *stdaudioin*, or any unlocked audio input driver, or file pointer if it is known. If used with the analog input driver AUIADC (Chapter 7.4.1), AUINPUT can also be used to configure the input channels' multiplexers.

If called without any command line arguments that change a value, AUINPUT will display the status of the audio driver as shown below

```
S:>auinput
stdaudioin:      0x203a, auii2ss::audioFile=3171(0xc63)
  ->Identify():  0x3b4f, auxsyncs::Identify returns "AUXSYNCS"
  ->op:          0x2041, auii2ss::audioFileOps=0(0x0)
    ->Ioctl():   0x3992, auxsyncs::AudioIoctl
    ->Read():    0x38cf, auii2ss::AudioRead
Sample rate:     48000
Bits per sample: 16
Buffer size:     512 16-bit words (256 16-bit stereo samples)
Buffer fill:     508 16-bit words (254 16-bit stereo samples)
```

```
Sample counter:  235803492
Overflows:       123022
```

In this example, slave audio synchronization driver AUXSYNCS.DL3 (Chapter 7.7.1) has been loaded on top of AUII2SS.DL3 (Chapter 7.5.5), replacing two of its methods, Identify() and Ioctl().

## 9.2 Control Program AUOUTPUT.DL3

```
Usage: AuOutput [-ddrv|-pfp|-rrate|-bbits|-sbufSize|-lvol|-v|+v|-h]
-ddrv   Connect to audio driver DRV.DL3
-pfp    Set output audio file pointer to fp (use with caution!)
-rrate  Set sample rate to rate
-bbits  Number of bits (16 or 32)
-sbufSz Set buffer size to bufSz 16-bit words
-lvol   Volume Level of maximum (vol = -128 .. 127.5)
-v|+v   Verbose on|off
-h      Show this help
```

AUOUTPUT lets the user display control several parameters of *stdaudioout*, or any unlocked audio input driver, or file pointer if it is known.

If called without any command line arguments that change a value, AUOUTPUT will display the status of the audio driver as shown below

```
S:>auoutput
stdaudioout:     0x1fea, auodac::audioFile=3139(0xc43)
  ->Identify():  0x3b4f, auxsyncs::Identify returns "AUXSYNCS"
  ->op:          0x1ff1, auodac::audioFileOps=0(0x0)
    ->Ioctl():   0x355b, auodac::AudioIoctl
    ->Write():   0x39fb, auxsyncs::AudioWrite
Sample rate:     47793
Bits per sample: 16
Buffer size:     4096 16-bit words (2048 16-bit stereo samples)
Buffer fill:     4 16-bit words (2 16-bit stereo samples)
Sample counter:  235977115
Underflows:      177796
Volume:          +0.0 dB of maximum level
```

In this example, slave audio synchronization driver AUXSYNCS.DL3 (Chapter 7.7.1) has been loaded on top of AUODAC.DL3 (Chapter 7.2.1), replacing two of its methods, Identify() and Write().

Note: To display symbol information, AUINPUT requires library TRACE.DL3.

# 10 Configuration Examples

Here are some configuration examples for loading different audio drivers.

For full options for each of these programs, have a look at the README.TXT / PDF file for each of the drivers.

## 10.1 Minimal config.sys for Playback

```
# New 2015 audio DAC out driver
AUODAC s
```

## 10.2 config.sys for Playback with Bass/Treble Controls and I2S + S/PDIF Outputs

```
# New 2015 audio DAC out driver
AUODAC s
# I2S automatic out; automatic is hardware, so doesn't require CPU
AUOI2SMA
# S/PDIF automatic out, parameter can be either 48000 or 96000 (default)
# If "v" is defined, stdaudioout volume control is copied to S/PDIF,
# otherwise it needs to be controlled manually (otherwise it stays at
# maximum level)
AUOSPDA 96000 v
# Equalizer, set 100 and 10000 Hz to +6 dB with Q Factor 0.7
FTOEQU
RUN SETEQU 1 3 100 +6 0.7
RUN SETEQU 2 3 10000 -6 0.7
```

## 10.3 Basic config.sys for Recording

```
# New 2015 audio DAC out driver
AUODAC s
# New 2015 audio ADC in driver
AUIADC s 48000 line1_1 line1_3
# DC Block; use at least this with analog input even if not using AGC
FTIDCBL
```

## 10.4 Versatile config.sys for Recording with AGC and I2S + S/PDIF Outputs

```
# New 2015 audio DAC out driver
AUODAC s
# New 2015 audio ADC in driver
AUIADC s 48000 line1_1 line1_3
# I2S automatic out; automatic is hardware, so doesn't require CPU
AUOI2SMA
# S/PDIF automatic out, parameter can be either 48000 or 96000 (default)
# If "v" is defined, stdaudioout volume control is copied to S/PDIF,
# otherwise it needs to be controlled manually (otherwise it stays at
# maximum level)
AUOSPDA 96000 v
# DC Block + AGC unit to stdaudioin
FTIAGC
```

## 10.5 config.sys for Playback/Recording from I2S in Slave Mode, and Monitoring to DAC with Automatic Synchronization

```
# Load I2S Slave Input driver and make it stdaudioin
AUII2SS s
# Load Line Out / Earphone output driver and make it stdaudioout
AUODAC s
# Connect and synchronize stdaudioout with stdaudioin slave
AUXSYNCS
# Copy stdaudioin to stdaudioout
# If loaded 'q' parameter, buffer under-/overflows are NOT reported
AUXPLAY
```

## 10.6 Loading/Unloading Drivers Using the VSOS Shell

Using the VSOS Shell Environment, you can use the DRIVER.DL3 program to load drivers to memory, and to later unload them.

If possible, you should always unload drivers in the reverse order of loading them. This is particularly true with drivers that connect to other drivers, like AUXSYNCS which connects to both the *stdaudioin* and *stdaudioout* drivers (in this case AUII2SS and AUODAC, respectively), and AUXPLAY which also uses *stdaudioin* and *stdaudioout*

Example: to load the drivers in Chapter 10.5, run the following commands:

```
S:>driver +auii2ss s
S:>driver +auodac s
S:>driver +auxsyncs
S:>driver +auxplay
```

To unload the drivers, enter the following commands:

```
S:>driver -auxplay
S:>driver -auxsyncs
S:>driver -auodac
S:>driver -auii2ss
```

# 11 VSOS Audio ioctl() Controls

VSOS Audio Drivers can be controlled from C language using ioctl() controls declared in <aucommon.h>.

There are many more definitions in the #include file <aucommon.h>. Refer to the documentation of the specific drivers you use for exact details on what of these functions they support and how to get access to a file pointer for that driver.

The ioctl() function prototype is

```
s_int16 ioctl(void *p, register int request, register char *arg);
```

where `p` is the file or device pointer (e.g. `stdaudioin` or `stdaudioout`), `request` is the type of the request, and `arg` is the optional argument.

ioctl() returns `S_ERROR` (-1) for an error (there was an error in the parameters, or the ioctl() for the `request` doesn't exist in this driver), any other value for success.

Generally, for functions that set a value, if `arg` is a pointer or a 16-bit calue, it is casted to c `char *` and passed to the function (e.g. IOCTL_AUDIO_SET_BITS in Chapter 11.2.5). If `arg` is a larger entity (e.g. 32-bit number), a pointer to the value is passed instead (e.g. IOCTL_AUDIO_SET_ORATE in Chapter 11.2.3).

Again, generally, for functions that return a 16-bit value where `S_ERROR` (-1) isn't included in the valid value range, the value is returned directly (e.g. IOCTL_AUDIO_GET_BITS in Chapter 11.2.4). Otherwise, the user needs to transmit a pointer to the return value in `arg` (e.g. IOCTL_AUDIO_GET_ORATE in Chapter 11.2.2). Not that in both cases ioctl() returns `S_ERROR` (-1) if there was an error in the call.

## 11.1 Resetting a Driver

### 11.1.1 IOCTL_RESTART

Restart driver. Normally this needs never be done.

Example:

```
ioctl(fp, IOCTL_RESTART, NULL);
```

## 11.2 Controlling Sample Rate and Bit Width

### 11.2.1 IOCTL_AUDIO_SET_RATE_AND_BITS

Set sample rate and number of bits. This is the recommended way of setting the sample rate and bit width with drivers like e.g I2S where there is a limit to sample rate and bit width combinations. Note that the sample rate / bit width argument may be larger than what can be fit into 16 bits, so it needs to be passed through a pointer.

Some drivers have very restricted number of sample rates supported. If you want to see what sample rate actually was set by the hardware, it is recommended to do a IOCTL_AUDIO_GET_IRATE or IOCTL_AUDIO_GET_ORATE call to see what you actually got.

- labs(rateBits) = sampleRate, may be in fractional sample rate format (Chapter 11.6).
- if rateBits < 0, then use 32-bit I/O
- Sets both input and output sample rate, if applicable
- Not available with Slave Mode drivers

Example:

```
s_int32 rateBits = -48000; /* Set to 48000 Hz, 32 bits */
if (ioctl(fp, IOCTL_AUDIO_SET_RATE_AND_BITS, (char *)(&rateBits))) {
  printf("Couldn't set sample rate and bits\n");
}
```

### 11.2.2 IOCTL_AUDIO_GET_IRATE, IOCTL_AUDIO_GET_ORATE

Get integer part of the current sample rate. Note that sample rate may be larger than what can fit into 16 bits, so it needs to be passed through a 32-bit pointer.

Some drivers have very restricted number of sample rates supported. If you want to see what sample rate actually was set by the hardware, it is recommended to do a IOCTL_AUDIO_GET_IRATE or IOCTL_AUDIO_GET_ORATE call to see what you actually got.

- Not available with Slave Mode drivers

Example for driver with input:

```
s_int32 sampleRate;
if (ioctl(fp, IOCTL_AUDIO_GET_IRATE, (char *)(&sampleRate))) {
  printf("Couldn't get sample rate\n");
}
```

Example for driver with output:

```
s_int32 sampleRate;
if (ioctl(fp, IOCTL_AUDIO_GET_ORATE, (char *)(&sampleRate))) {
  printf("Couldn't get sample rate\n");
}
```

### 11.2.3 IOCTL_AUDIO_SET_IRATE, IOCTL_AUDIO_SET_ORATE

Set sample rate, which may be in fractional sample rate format (Chapter 11.6). Note that sample rate may be larger than what can fit into 16 bits, so it needs to be passed through a 32-bit pointer.

- Only for Master Mode drivers
- It is recommended to use IOCTL_AUDIO_SET_RATE_AND_BITS instead

Example for driver with input:

```
s_int32 sampleRate = 48000;
if (ioctl(fp, IOCTL_AUDIO_SET_IRATE, (char *)(&sampleRate))) {
  printf("Couldn't set sample rate\n");
}
```

Example for driver with output:

```
s_int32 sampleRate = 48000;
if (ioctl(fp, IOCTL_AUDIO_SET_ORATE, (char *)(&sampleRate))) {
  printf("Couldn't set sample rate\n");
}
```

### 11.2.4 IOCTL_AUDIO_GET_BITS

Get number of bits for driver.

Example:

```
bits = ioctl(fp, IOCTL_AUDIO_GET_BITS, NULL);
```

### 11.2.5 IOCTL_AUDIO_SET_BITS

Set number of bits for driver.

Example:

- bits may be 16 or 32
- With Master Mode drivers it is recommended to use IOCTL_AUDIO_SET_RATE_AND_BITS instead

Example:

```
if (ioctl(fp, IOCTL_AUDIO_SET_BITS, (char *)(32))) {
  printf("Couldn't set bits\n");
}
```

## 11.3 Controlling Audio Buffers

### 11.3.1 IOCTL_AUDIO_GET_INPUT_BUFFER_FILL

Get input buffer fill state in 16-bit words.

- Only for drivers with input capability

Example:

```
iBufFill = ioctl(fp, IOCTL_AUDIO_GET_INPUT_BUFFER_FILL, NULL);
```

### 11.3.2 IOCTL_AUDIO_GET_INPUT_BUFFER_SIZE

Get input buffer size in 16-bit words.

- Only for drivers with input capability

Example:

```
iBufSize = ioctl(fp, IOCTL_AUDIO_GET_INPUT_BUFFER_SIZE, NULL);
```

### 11.3.3 IOCTL_AUDIO_SET_INPUT_BUFFER_SIZE

Set input buffer size in 16-bit words.

- Only for drivers with input capability

Example:

```
if (ioctl(fp, IOCTL_AUDIO_SET_INPUT_BUFFER_SIZE, (char *)(1024))) {
  printf("Couldn't set input buffer size\n");
}
```

### 11.3.4 IOCTL_AUDIO_GET_OUTPUT_BUFFER_FREE

Get how many 16-bit words there are free in the output buffer.

- Only for drivers with DSP output capability

Example:

```
iBufFill = ioctl(fp, IOCTL_AUDIO_GET_OUTPUT_BUFFER_FREE, NULL);
```

### 11.3.5 IOCTL_AUDIO_GET_OUTPUT_BUFFER_SIZE

Get output buffer size in 16-bit words.

- Only for drivers with DSP output capability

Example:

```
oBufSize = ioctl(fp, IOCTL_AUDIO_GET_OUTPUT_BUFFER_SIZE, NULL);
```

### 11.3.6 IOCTL_AUDIO_SET_OUTPUT_BUFFER_SIZE

Set output buffer size in 16-bit words.

- Only for drivers with DSP output capability

Example:

```
if (ioctl(fp, IOCTL_AUDIO_SET_OUTPUT_BUFFER_SIZE, (char *)(1024))) {
  printf("Couldn't set output buffer size\n");
}
```

## 11.4 Volume Control

### 11.4.1 IOCTL_AUDIO_GET_VOLUME

Get volume. Volume is a number between 0 - 511 where 256 is full-scale, and each successive number represents a volume gain step of -0.5 dB. See table below:

| IOCTL_AUDIO_GET_VOLUME argument table | | |
|---|---|---|
| **Argument** | **Amplification** | **Description** |
| 0 | +128.0 dB | Insane amplification |
| 1 | +127.5 dB | Insane amplification minus 0.5 dB |
| ... | ... | ... |
| 255 | +0.5 dB | Slightly louder than full-scale volume |
| 256 | 0.0 dB | Full-scale volume |
| 257 | -0.5 dB | Almost full-scale volume |
| ... | ... | ... |
| 509 | -126.0 dB | Very silent |
| 510 | -∞ dB | No sound, may not turn off driver |
| 511 | -∞ dB | No sound, may turn off driver |

A driver may limit the range it actually accepts for its volume settings. E.g. the analog output driver AUODAC only supports the range between 256 (0.0 dB) and 511 (analog driver power-down). As another example, the S/PDIF driver supports the range between 208 (+24.0 dB) and 511 (silence). If a driver does not support the whole range, it will automatically limit itself so you can still call it with the extreme values.

511 is a special value that allows e.g. the audio driver to turn itself off (supported by e.g. AUODAC). Use with caution!

Example:

```
volume = ioctl(fp, IOCTL_AUDIO_GET_VOLUME, NULL);
```

### 11.4.2 IOCTL_AUDIO_SET_VOLUME

Set volume. Scale for volume is the same as for IOCTL_AUDIO_GET_VOLUME (Chapter 11.4.1).

Example:

```
/* Set full scale volume */
if (ioctl(fp, IOCTL_AUDIO_SET_VOLUME, (char *)(256))) {
  printf("Couldn't set volume\n");
}
```

## 11.5 Miscellaneous Controls

### 11.5.1 IOCTL_AUDIO_GET_SAMPLE_COUNTER

Get sample counter. This value may be used to synchronize input and output (e.g. by the driver AUXSYNCS, Chapter 7.7.1).

Example:

```
s_int32 sampleCounter;
if (ioctl(fp, IOCTL_AUDIO_GET_SAMPLE_COUNTER, (char *)(&sampleCounter))) {
  printf("Couldn't get sample counter\n");
}
```

### 11.5.2 IOCTL_AUDIO_GET_OVERFLOWS

Get overflow sample counter for the input buffer.

If this number changes while an audio input program is running, this is an indication of a program performance or input/output buffer size issue.

If nobody cosumes samples from the input audio driver, this value increases at the rate of the sample counter that can be read with IOCTL_AUDIO_GET_SAMPLE_COUNTER.

- Only for drivers with input

Example:

```
s_int32 overFlow;
if (ioctl(fp, IOCTL_AUDIO_GET_OVERFLOWS, (char *)(&overFlow))) {
  printf("Couldn't get overflow counter\n");
}
```

### 11.5.3 IOCTL_AUDIO_GET_UNDERFLOWS

Get underflow sample counter for the output buffer.

If this number changes while an audio output program is running, this is an indication of a program performance or input/output buffer size issue.

If nobody produces samples for the output audio driver, this value increases at the rate of the sample counter that can be read with IOCTL_AUDIO_GET_SAMPLE_COUNTER.

- Only for drivers with output

Example:

```
s_int32 underFlow;
if (ioctl(fp, IOCTL_AUDIO_GET_UNDERFLOWS, (char *)(&underFlow))) {
  printf("Couldn't get underflow counter\n");
}
```

### 11.5.4 IOCTL_AUDIO_SELECT_INPUT

Select analog input. Parameter bitmask must have one stereo element, or one left and one right element. Definitions can be found in <aucommon.h>

Stereo elements:

- AID_FM

Left elements:

- AID_LINE1_1, AID_LINE3_1, AID_LINE2_1, AID_MIC1, AID_DIA1

Right elements:

- AID_LINE1_2, AID_LINE3_2, AID_LINE2_2, AID_MIC2, AID_DIA2, AID_DIA3, AID_LINE1_3

- This ioctl() is only applicable for the AUODAC driver.
- Optionally, AID_DEC6 may also be defined. It activates the high-quality down-by-6 decimator.

Example:

```
s_int32 sampleCounter;
if (ioctl(fp,IOCTL_AUDIO_SELECT_INPUT,(char *)(AID_LINE1_1|AID_LINE1_3))) {
  printf("Couldn't select input\n");
}
```

### 11.6 Fractional Sample Rates

For most audio drivers, setting the sample rate with an accuracy of 1 Hz is enough. However, some drivers are internally capable of setting the sample rate with better accuracy. As an example the analog output driver AUODAC.DL3 can set the sample rate with an accuracy of approximately 0.09 Hz. This makes a driver more useful when streaming audio in slave mode, e.g. when using the AUXSYNCS.DL3 synchronization driver.

Because being able to set the sample rate with higher than 1 Hz accuracy was a new VSOS feature for 2016, it was important to maintain compatibility with older software that is incapable of setting the sample rate with sub-hertz accuracy.

To set a fractional sample rate, ioctl()'s IOCTL_AUDIO_SET_RATE_AND_BITS, IOCTL_AUDIO_SET_IRATE, and IOCTL_AUDIO_SET_ORATE all can take their parameters in a 32-bit integer-compatible fractional representation, where bits 30:24 of the sample rate parameter represent 1/128 Hz increments, as shown in the following table.

| 32-bit fractional sample rate representation | | |
|---|---|---|
| **Bits** | **Range** | **Description** |
| 31 | 0 | Not used, set to 0 |
| 30:24 | 0..127 | Sample rate fractions in 1/128 Hz |
| 23:0 | 0..16777215 | Sample rate integer part in Hz |

| 32-bit fractional sample rate examples | |
|---|---|
| **Sample rate** | **Representation** |
| 44100 Hz | 0x0000ac44 |
| $44100\frac{1}{128}$ Hz | 0x0100ac44 |
| 44100.5 Hz | 0x4000ac44 |
| $44100\frac{127}{128}$ Hz | 0x7f00ac44 |

Audio drivers which are not interested in the sample rate's fractional part, mask away bits 30:24 from ioctl() sample rate setting parameters.

To maintain compatilibity with software unaware of the fractional sample rate presentation format, IOCTL_AUDIO_GET_IRATE and IOCTL_AUDIO_GET_ORATE only return the integer portion of the sample rate.

# 12 Controlling Audio from VSOS Shell with UiMessages

When using the VSOS Shell, some audio functions may be controlled even if running a VSOS program that doesn't take audio controls. If the TTY is not in RAW mode, the following escape sequences defined in <uimessages.h> may be sent to the shell.

## 12.1 Setting Volume anywhere from VSOS Shell

Note that <B> here means sending ASCII code 1, invoked in most terminal emulation programs by pushing Ctrl-B.

Volume up by 1/2 dB:
<B>111ms

Volume down by 1/2 dB:
<B>112ms

Set attenuation to -HH/2 dB, where HH is a hexadecimal number:
<B>206mHHs

Example:
To set volume to -20 dB, you need to send 40 = 0x28:
<B>206m28s

## 12.2 Sending Equalizer Controls from VSOS Shell

The filters are accessed with UiMessages that have the following format, where X is the filter number (0..f), and YY is the 16-bit signed value presented as an unsigned 16-bit hexadecimal number. <B>21XmYYs

Example:
Let's assume we have the following configuration lines in config.txt:

```
RUN SETEQU 1 3 100 0 0.7
RUN SETEQU 2 3 10000 0 0.7
```

Now, to set bass (filter channel 1) to +6 dB (6), send the following command:
<B>210m6s

To set treble (filter channel 2) to -12 dB (0xfff4), send the following command:
<B>211mfff4s

Up To 16 channels may be accesses with messages ranging from 0x210 to 0x21f.

# 13 Audio Decoders

Library audiodec automatically chooses between many Audio Decoders when presented an audio file. The libraries, their respective decode audio formats, and their clock rate requirements, are presented below.

NOTE: The clock speeds have been tested with a Class 4 SD card and “typical test files”. While VLSI believes the information to be accurate, clock rates should still be interpreted as estimations. The estimations are given for a system with a typical interrupt load and with a 8 KiW audio output buffer. An example alternative configuration file that sets the system up for best playback audio performance is provided in file config_audio_decoders.txt in the *VSOS Root and Libraries Source Code* package.

| Audio Decoders | | |
|---|---|---|
| **LibName** | **Format** | **Description** |
| decaac | AAC | AAC in ADTS and MP4 containers (.aac, .m4a, .mp4, .3gpp) |
| decaiff | AIFF | Apple uncompressed PCM format |
| decalac | ALAC | Apple lossless in MP4 (.mp4, .m4a) or CAFF (.caf) container |
| decalac | APE | Monkey’s audio |
| decdsd | DSD | DSD bitstream files in .DSF and .DFF container, LSb first only |
| decflac | FLAC | Free Lossless Audio Codec |
| decmp3 | MP3 | MPEG audio layer 3 |
| decvorb | Ogg Vorbis | Vorbis audio in Ogg container |
| decwav | RIFF WAV | Many RIFF WAV subformats |
| decwma | WMA | Windows Media Audio |
| mp4file | - | Determines if MP4 file contains ALAC or AAC |

NOTE: For formats where there may be more than 2 audio channels, only files up to 2 audio channels are supported.

| Library decaac subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 30 MHz | AAC up to 48 kHz, 280 kbit/s |

| Library decaiff subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 12 MHz | AIFF up to 96 kHz 16-bit |
| 18 MHz | AIFF up to 96 kHz 24-bit |
| 37 MHz | AIFF up to 192 kHz 24-bit |
| 61 MHz | AIFF up to 352 kHz 24-bit |

| Library decalac subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 37 MHz | Apple lossless up to 48 kHz 16-bit |
| 74 MHz | Apple lossless up to 96 kHz 24-bit |

| Library decape subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 61 MHz | Monkey's Audio up to 48 kHz 24-bit, profile Fast |
| 67 MHz | Monkey's Audio up to 48 kHz 24-bit, profile Normal |
| 79 MHz | Monkey's Audio up to 48 kHz 24-bit, profile High |
| N/A | Monkey's Audio, profiles Extra High and Insane |

| Library decdsd subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 49 MHz | DSD64 (2.8 MHz, 1-bit) |
| 86 MHz | DSD128 (5.6 MHz, 1-bit) |
| 92 MHz | DSD256 (11.3 MHz, 1-bit) |

| Library decflac subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 12 MHz | FLAC up to 16 kHz, 16-bit |
| 18 MHz | FLAC up to 32 kHz, 16-bit |
| 25 MHz | FLAC up to 48 kHz, 16-bit |
| 37 MHz | FLAC up to 96 kHz, 16-bit |
| 43 MHz | FLAC up to 96 kHz, 24-bit |

| Library decmp3 subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 12 MHz | MP3 at 8 kHz, 8 kbit/s |
| 31 MHz | MP3 up to 48 kHz, 320 kbit/s |

| Library decvorb subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 12 MHz | Ogg Vorbis up to 16 kHz, 73 kbit/s |
| 18 MHz | Ogg Vorbis up to 32 kHz, 151 kbit/s |
| 37 MHz | Ogg Vorbis up to 48 kHz, 346 kbit/s |
| 55 MHz | Ogg Vorbis up to 96 kHz, 362 kbit/s |

| Library decwav subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 12 MHz | RIFF WAV up to 96 kHz 16-bit |
| 18 MHz | RIFF WAV up to 96 kHz 24-bit |
| 31 MHz | RIFF WAV up to 192 kHz 24-bit (e.g. DXD format) |
| 55 MHz | RIFF WAV up to 352 kHz 24-bit (e.g. DXD format) |
| 68 MHz | RIFF WAV up to 352 kHz 32-bit floating-point (e.g. DXD format) |

| Library decwma subformats and clock requirements | |
|---|---|
| **Clock** | **Description** |
| 61 MHz | All WMA files in VLSI Solution's internal test suite |

### 13.1 Decoder Loop Functionality

Some of the audio decoders include a chance to play a part of the audio file in a loop.

Depending on the decoder, there may or may not be support for the

The list of audio decoders that contains loop functionality, and the level of support, is provided in the following table:

| Audio decoders with loop functionality | | | | |
|---|---|---|---|---|
| **LibName** | **Set timing**[1] | **Sample accurate**[2] | **Smooth**[3] | **Comments** |
| decvorb | Yes | Yes | No | - |
| decwav | Yes | Yes[4] | No | - |

[1] If this feature is not available, the decoder is only able to loop the complete audio file. To make sure user software is compatible with potential future versions of the driver which may start supporting the Set Timing feature, Loop structure should be set as follows:

```
loop->startSeconds = loop->endSeconds = loop->endSamples = 0;
loop->endSeconds = 0xFFFFFFFFU;
```

[2] If this feature is available, looping is sample-accurate. If not available, loop start and stop points may vary slightly.

[3] If this feature is available, loop supports the CFL_DECLICK flag which declicks the loop but is not sample accurate. If this feature is missing from the decoder, flag CFL_DECLICK is ignored.

[4] Exception: IMA ADPCM is not sample accurate.

An example of how to use the loop feature is provided in solution PlayFileLoop in the *VSOS Root and Libraries Source Code* package. Read the README.TXT file for details.

# 14 Audio Encoders

There are currently two Audio Encoders. They are for Ogg Vorbis and MP3 formats. Both are intended to operate properly at the 60 MHz standard operating speed. For details of how to use them, see the source code for the VSIDE solution Rec (more information for Rec is available in the *VSOS Shell document*).

## 14.1 ENCVORB.DL3 - Ogg Vorbis Encoder

ENCVORB.DL3 offers a high-quality encoder for the free end open Ogg Vorbis audio format. This is the recommended format for those users that don't specifically need the MP3 format, and for whom the variable bitrate property of Ogg Vorbis is not a problem.

The Ogg Vorbis encoder is a Variable BitRate encoder, and works best when given an encoding quality value. The range for the quality is from 0 and 10 (10 is the best; qualities above 6 may require a higher clock speed than 60 MHz). If set to Constant bitrate, the encoder will convert that to an approximate quality value, and still use Variable BitRate.

## 14.2 ENCMP3.DL3 - MP3 Encoder (VS1205 only)

ENCMP3.DL3 offers a high-quality encoder for the popular MP3 audio format. While not capable of getting quite the same quality as Ogg Vorbis when using similar bit-rates, the MP3 encoder still offers high fidelity sound at bit-rates between 160-192 kbit/s, and acceptable speech quality at very much lower bitrates.

The MP3 encoder can operate either with a Variable BitRate or Constant BitRate encoder depending on whether a quality or bitrate value is set. For best quality per bit, Variable BitRate is recommended.

## 14.3 ENCFLAC.DL3 - FLAC Encoder

ENCFLAC.DL3 offers a lossless encoder for 16-bit audio up to 48 kHz. Typically a 48 kHz 16-bit stream is compressed from 1.536 Mbit/s to 0.8-1.3 Mbit/s. Because encoding is lossless, compression efficiency depends on the audio data.

Because the bitrates are much higher than for the Ogg Vorbis or MP3 encoders, storing the result to e.g. an SD card may require a separate buffer memory.

# 15 Latest Document Version Changes

This chapter describes the latest changes to this document.

### Version 3.60, 2020-10-13

- Better explanation of sample rate availability for driver AUODAC.DL3 in Chapter 7.2.1, *Driver AUODAC.DL3*.
- Added mention of new volume control for driver AUOOSET.DL3 in Chapter 7.3.1, *Driver AUOOSET.DL3*.
- Other, minor modifications and typo corrections.

### Version 3.58, 2019-08-24

- Added mention that all S/PDIF drivers are incompatible with USB in Chapter 7.6, *S/PDIF Audio Drivers*.
- Replaced incorrect text IOCTL_AUDIO_GET_OUTPUT_BUFFER_SIZE with IOCTL_SET_OUTPUT_BUFFER_SIZE in Chapter 11.3.6, *IOCTL_SET_OUTPUT_BUFFER_SIZE*.

### Version 3.57, 2019-04-10

Minor corrections, updated for VSOS 3.57.

### Version 3.55b, 2018-06-12

Added Chapter 8.5, *Noise Killer Drivers*.

### Version 3.55a, 2018-04-05

- Added new first reflection flag “-f” to Reverb Generators in Chapter 8.4, *Reverb Generator Audio Drivers*.
- Added another Reverb Generator Driver and Chapter 8.4.2, *Driver FTOREV23*.

### Version 3.55, 2018-04-05

Added Chapter 8.4, *Reverb Generator Audio Drivers*.

### Version 3.52, 2018-01-22

Release for VSOS 3.52, no major changes.

### Version 3.42, 2017-05-18

Release for VSOS 3.42.

- Added several new S/PDIF drivers to Chapter 7.6, *S/PDIF Audio Drivers*.
- Added Chapter 14.3, *ENCFLAC.DL3 - FLAC Encoder*.
- Added Chapter 8.3, *Pitch Shifter / Speed Shifter Audio Drivers*.

### Version 3.40, 2016-11-03

Release for VSOS 3.40.

### Version 3.30a, 2016-07-14

This is a bug patch release for VSOS 3.30.

- Added comment for audio buffer size for speed measurements made in Chapter 14.
- Ctrl-A (ASCII code 1) change to Ctrl-B (ASCII code 2) for invoking UI messages in Chapter 12, *Controlling Audio from VSOS Shell with UiMessages* didn't work properly in release 3.30. Fixed.

### Version 3.30, 2016-06-22

This is a release for VSOS 3.30.

- Added Chapter 13, *Audio Decoders*, and Chapter 14, *Audio Encoders*.
- Added new HiRes audio decoders to the system, like DSD, ALAC and AIFF, and extended some decoders to the HiRes domain, like WAV and FLAC.
- Changed version numbering to reflect on the VSOS release this document is written for.
- Ctrl-A (ASCII code 1) changed to Ctrl-B (ASCII code 2) for invoking UI messages in Chapter 12, *Controlling Audio from VSOS Shell with UiMessages*.

# 16 Contact Information

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

URL: http://www.vlsi.fi/
Phone: +358-50-462-3200
Commercial e-mail: sales@vlsi.fi

For technical support or suggestions regarding this document, please participate at
http://www.vsdsp-forum.com/
For confidential technical discussions, contact
support@vlsi.fi