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**
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.

The document ends with Chapter 13, *Latest Document Version Changes*, and Chapter 14, *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

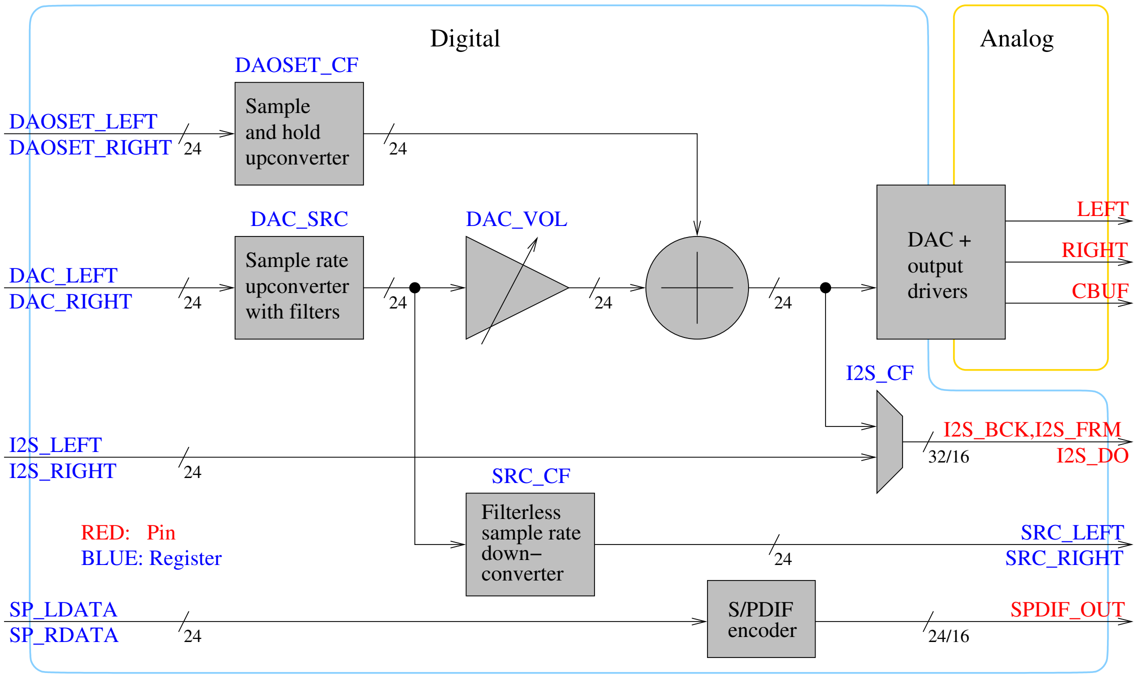

Figure 1: VS1005g playback (DA) audio paths

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

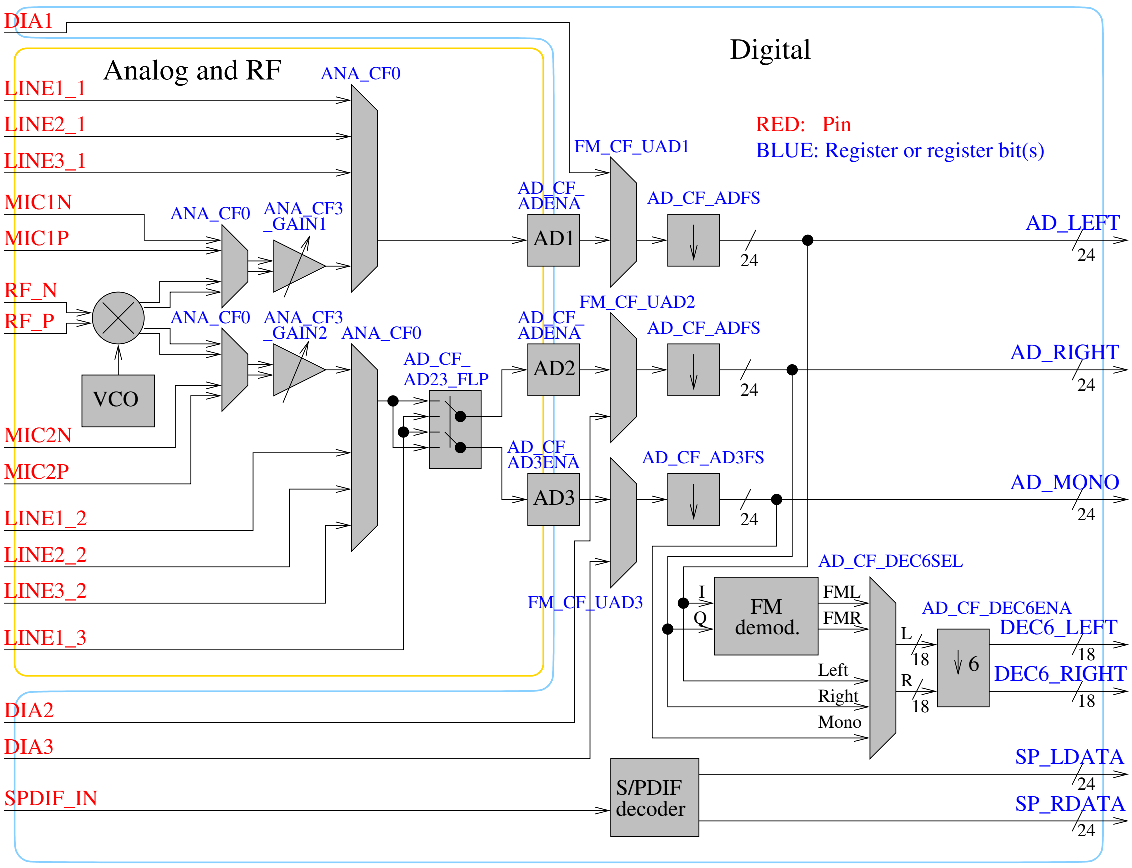

Figure 2: VS1005g recording (AD, FM, etc) signal paths

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

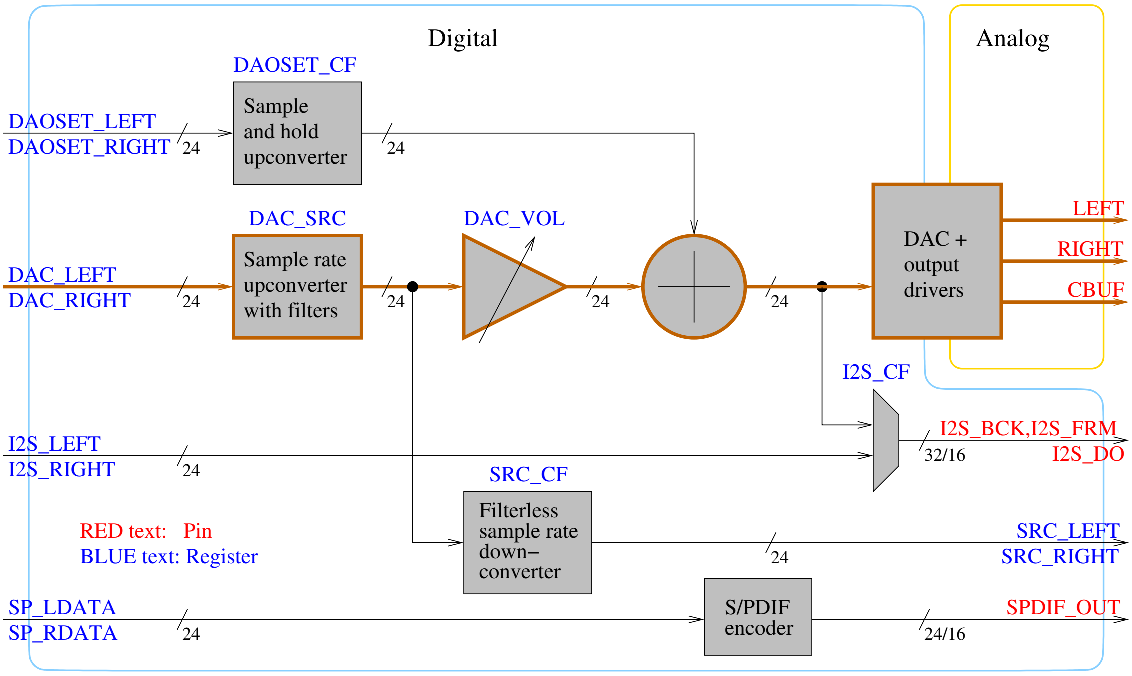

Figure 3: AUODAC.DL3 signal paths shown in **bold brown**

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 1 Hz accuracy upto 96000 Hz. Audio is upconverted to an extremely high rate of 6.144 MHz by a high-quality Sample rate up-converter.

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

7.3 Analog Side Path Audio Drivers

7.3.1 Driver AUOOSET.DL3

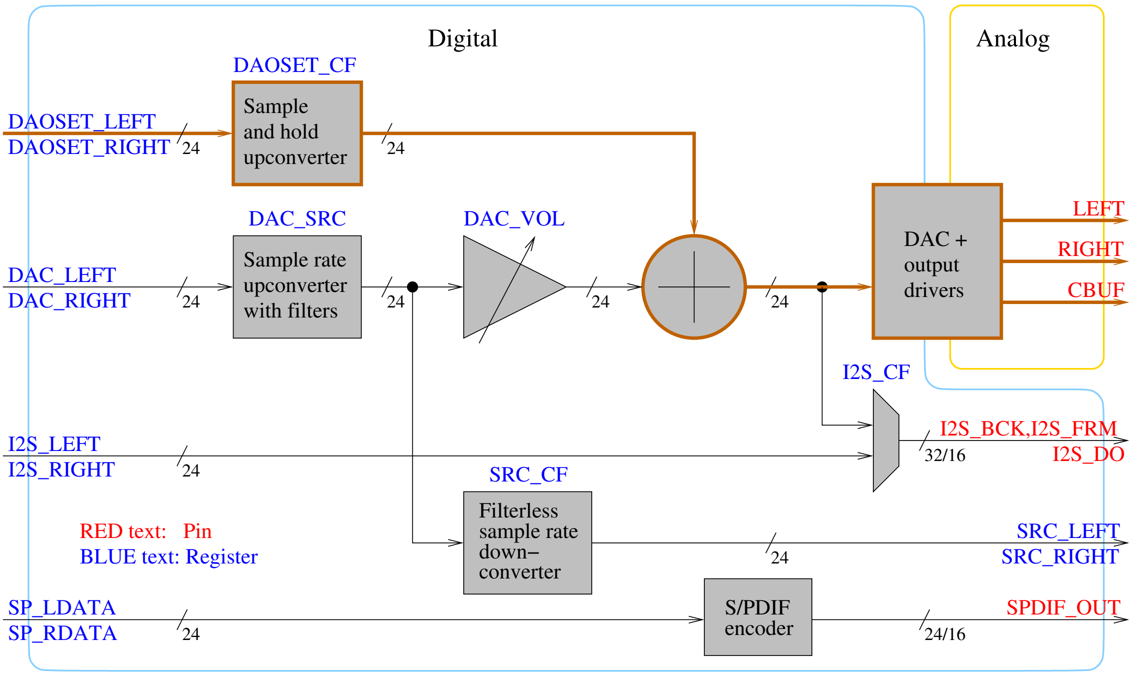

Figure 4: AUOOSET.DL3 signal paths shown in **bold brown**

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 upto 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 upto 150 Hz error. While not a problem for effects sounds, this may make accurate timing difficult.

7.4 Analog Input ADC Audio Drivers

7.4.1 Driver AUIADC.DL3

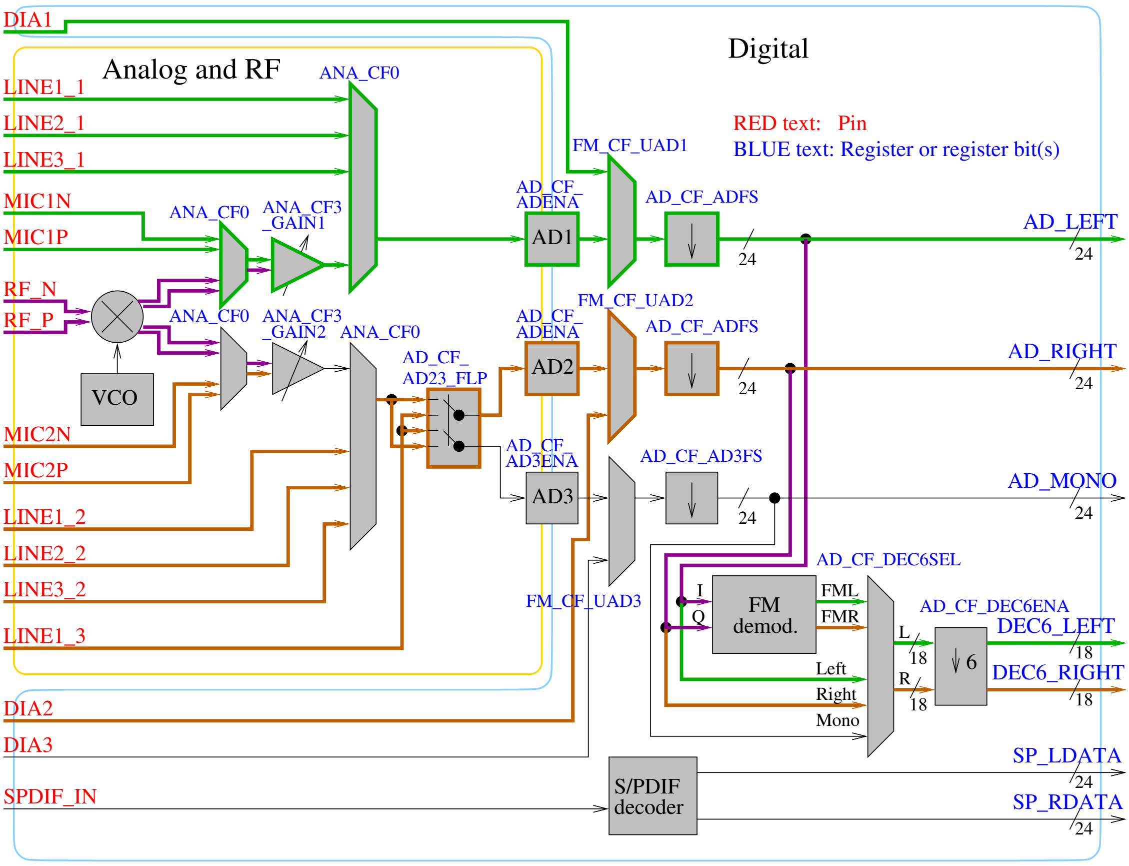

Figure 5: AUIADC.DL3 selectable input signal paths shown in **bold green** for the left channel, and **bold brown** for the right channel. Alternative RF audio path shown in **bold magenta**

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, and IOCTL_AUDIO_SET_ORATE. 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 *stdaudiout* if started with parameter “s”. Otherwise, the drivers need to be opened and accessed manually.

7.5.1 Driver AUOI2SMA.DL3

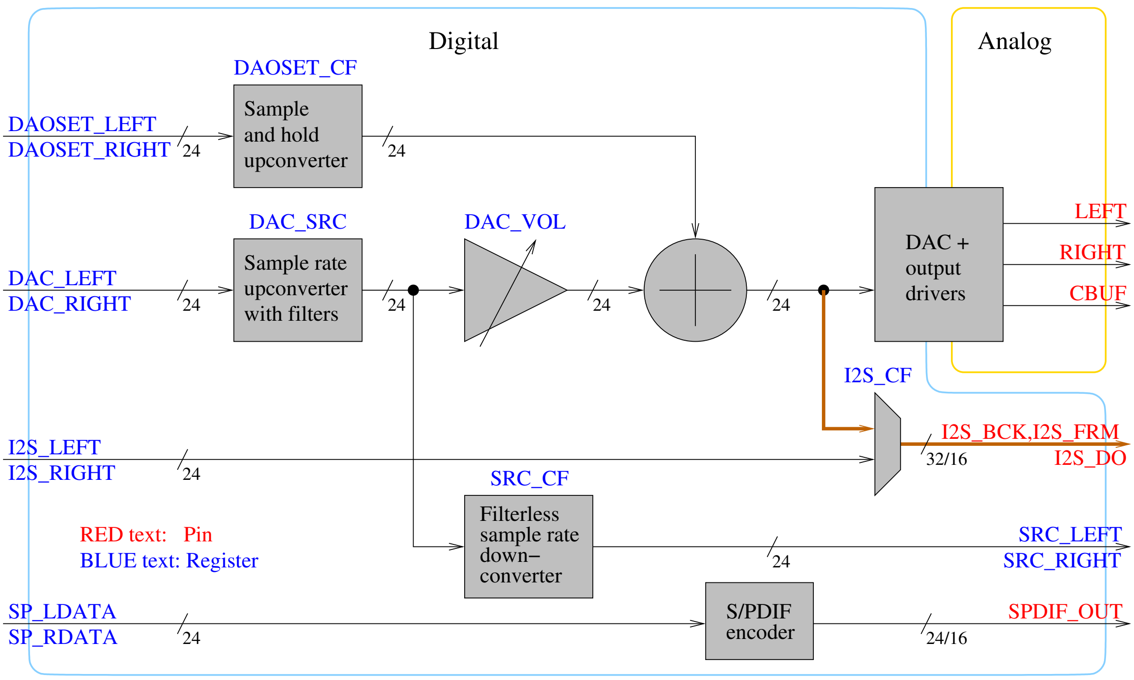

Figure 6: AUOI2SMA.DL3 audio path shown in **bold brown**

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

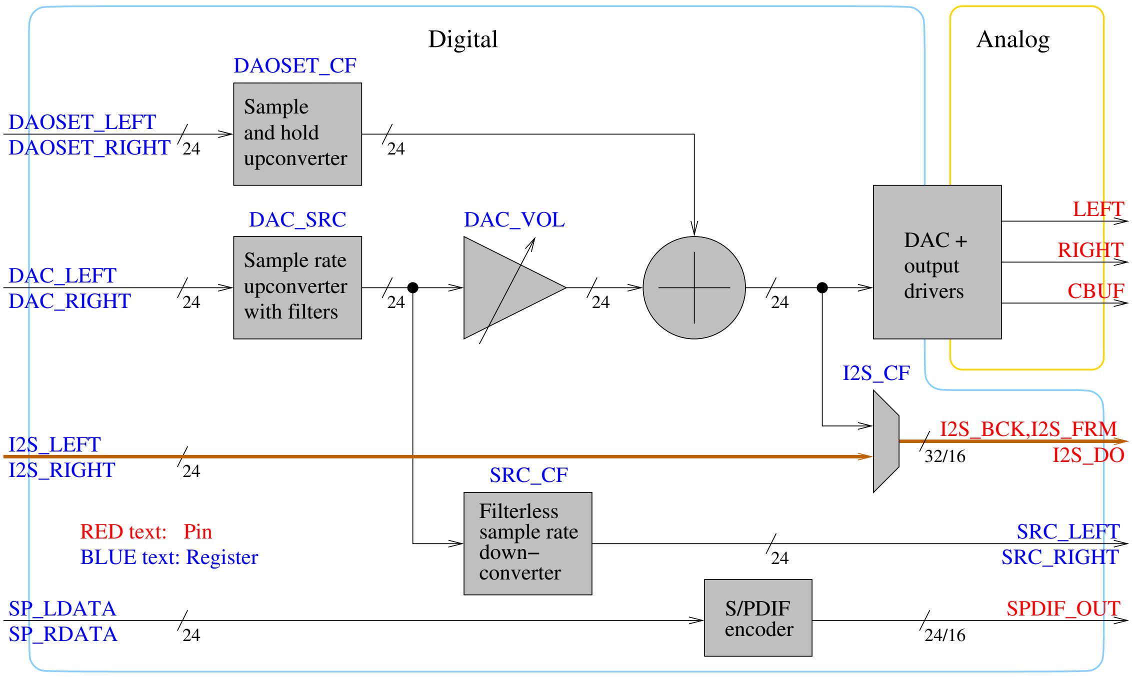

Figure 7: AUOI2SM.DL3 audio path shown in **bold brown**

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

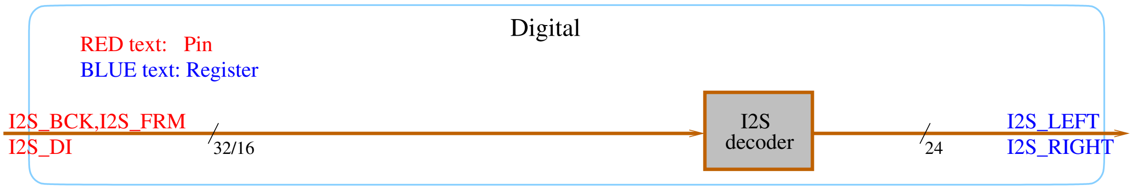

Figure 8: AUII2SM.DL3 audio path shown in **bold brown**

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

7.6.1 Driver AUOSPDA.DL3

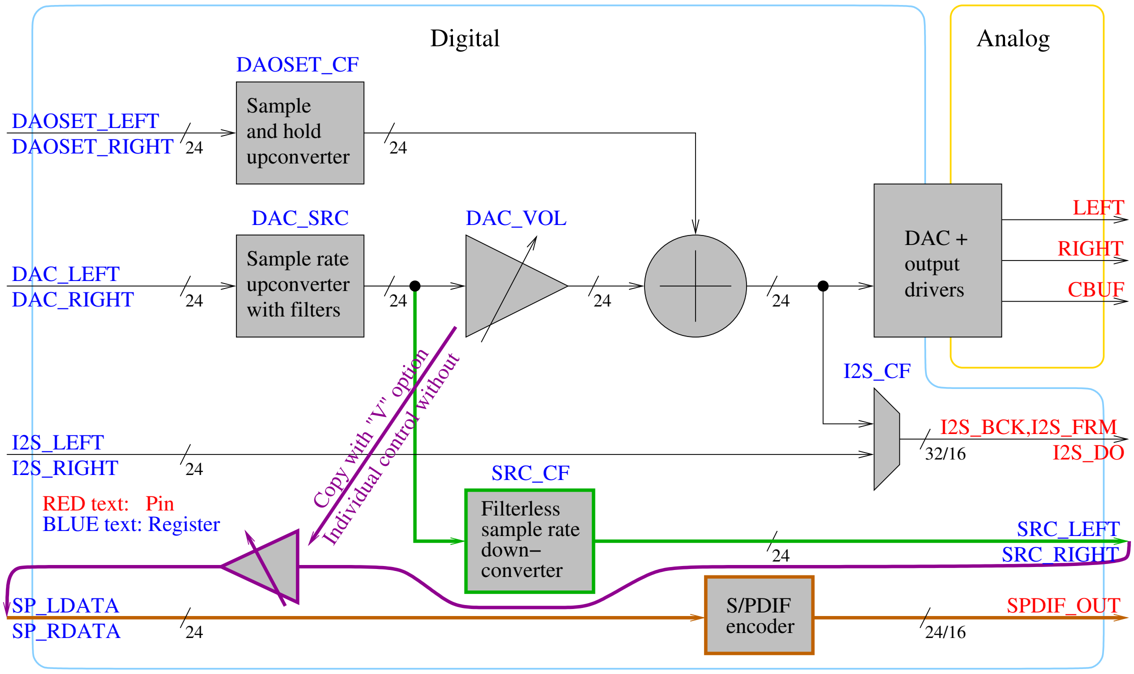

Figure 9: AUOSPDA.DL3 audio path shown in **bold brown**. Software driver connecting to the filterless sample rate converter (**bold green**) shown in **bold magenta**

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

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.

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
```

7.7 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 upto 97500 Hz, so standard sample rates upto 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 stdaudiouit 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

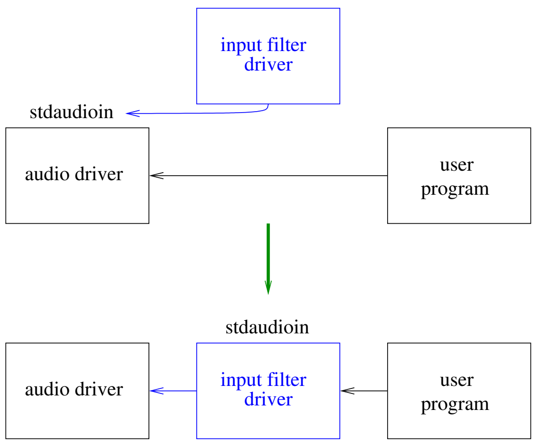

Figure 10: A filter input driver connects to the stdaudioin chain

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!

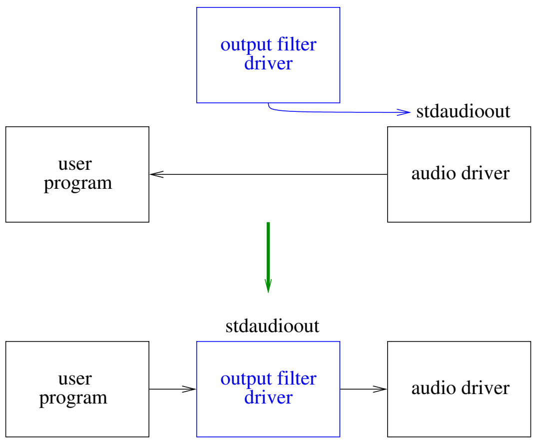

Figure 11: A filter output driver connects to the *stdaudioout* chain

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.

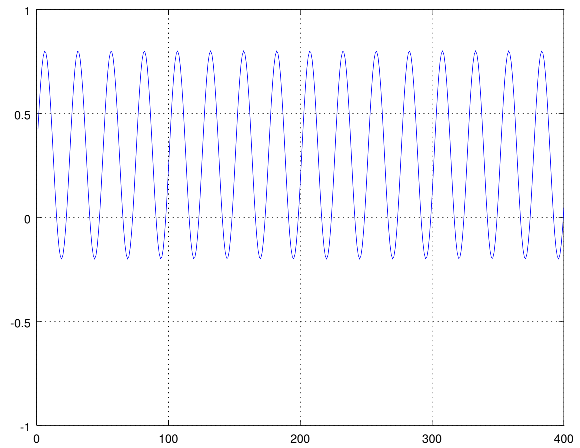

Figure 12: Audio with exaggerated DC offset

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.

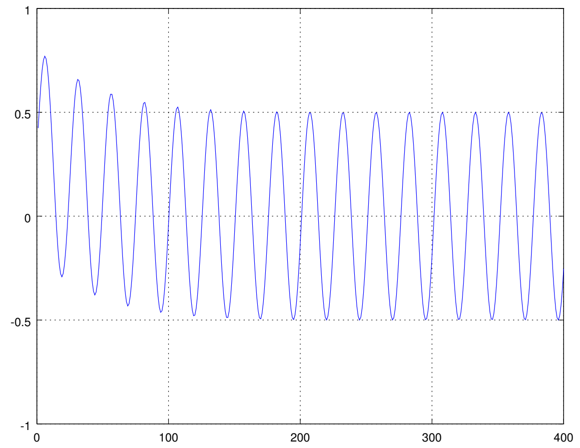

Figure 13: Audio with DC blocking

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.

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.

9 Audio Control Programs

These programs are useful for displaying and changing audio parameters, as well as debugging audio interfaces.

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
```

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)
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)
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 stdaudiouit 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 `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 *)(&sampleRate))) {
  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_GET_OUTPUT_BUFFER_SIZE

Set output buffer size in 16-bit words.

- Only for drivers with DSP output capability

Example:

```
if (ioctl(fp, IOCTL_AUDIO_GET_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 <A> here means sending ASCII code 1, invoked in most terminal emulation programs by pushing Ctrl-A.

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

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

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

Example:
To set volume to -20 dB, you need to send 40 = 0x28:
<A>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. <A>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:
<A>210m6s

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

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

13 Latest Document Version Changes

This chapter describes the most important changes to this document.

Version 1.03, 2016-02-15

This is a release for VSOS 3.26.

- AuInput and AuOutput output updated (Chapters 9.1 and 9.2).
- Added new concept, fractional sample rates (Chapter 11.6).
- Added new driver, AUXPLAY.DL3 (Chapter 7.8.1).

Version 1.02, 2016-01-27

- New application AUOUTPUT.DL3 (Chapter 9.2).
- New functions and modified interface to application AUINPUT.DL3 (Chapter 9.1).
- New slave audio synchronization driver AUXSYNCS (Chapter 7.7.1).
- New Chapter 10.6, *Loading/Unloading Drivers Using the VSOS Shell*.
- Clarified colour scheme for signal path Figures 3, 4, 5, 6, 7, 8, and 9.
- Typo corrections.

Version 1.01, 2015-09-14

Corrections, e.g. Figure 8.

Version 1.00, 2015-09-04

First release.

14 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