VS1005 APPNOTE: GETTING STARTED WITH DEVELOPER BOARD

Getting started with VS1005 developer board

Revision History

Rev.	Date	Author	Description
1.00	2016-06-23	HV	Initial version.

Contents

List of Figures

Listings

1 Introduction

Documentation available for the VS1005 Developer Board includes the VS1005 Datasheet, VS1005 VSOS Programmer's manual, VS1005 VSOS Audio Subsystem and VS_DSP[4] User's Manual documents. Also schematics of the board are available. The scope of those documents is to describe what is on the table. The target of this document is to help on the first experiences with the developer board.

This document describes procedure for getting development environment ready for VS1005 developer board, gives some examples and warns about some common pitfalls. The scope is very broad and all isues can't be covered thoroughly.

2 Definitions

~ Shortcut to user's home directory on Linux.

\ Folder separator on Microsoft Windows, shell escape character on Linux.

/ Directory separator on Linux, parameter separator on Microsoft Windows.

ADC Analog to digital converter.

AGC Automatic gain control.

DAC Digital to analog converter.

dB Decibel, a logarithmic unit that indicates the ratio of two powers. 1 bel, which equals 10 decibels, is a power ratio of 10.

Directory Hierarchical component of Linux filesystem

Folder Hierarchical component of Microsoft Windows filesystem

LSb Least significant bit.

LSB Least significant byte.

MSb Most significant bit.

MSB Most significant byte.

stdaudioin Default audio input in VSOS.

stdaudioout Default audio output in VSOS.

3 Connecting the Board

3.1 Required Hardware

When connecting developer board for application development, some hardware is required.

- VS1005 Developer Board
- VSIDE UART cable (Included in shipment)
- Mini-USB cable (Included in shipment)
- two free USB ports on computer

3.2 Making the Connections

Connect mini USB-B - USB-A cable to CN5 connector and free USB port on your PC. Connect VSIDE UART cable to CN9 header. Wire colors (R)ed, (G)reen, (Y)ellow and (B)lack are on the other side of the CN9 header and signals on the other. **DO NOT CONNECT RED WIRE TO 3V3 SIGNAL.** The other end of the VSIDE UART cable goes to your PC. Connecting red 5V connector to CN9 isn't required if USB cable to CN5 is connected.

Optionally connect 3.5mm stereo plug from audio source to line in connector CN3 and another 3.5mm stereo plug from line out connector CN2 to audio input of your amplifier. Headphones can be connected to the CN1 connector if line out isn't used. **DO NOT CONNECT ANYTHING ELSE BUT HEADPHONES TO CN1 CONNECTOR.** The virtual ground of headphones is around 1.2 V and shorting it to real ground ruins the output signal, and may even damage the VS1005.

To avoid ground loops, connect USB plugs to same panel, front of PC, rear of PC or USB hub but don't mix them. If signal source or external output amplifier has different ground potential, ground loop may generate noise mostly in mains power frequency. Use headphones and listen for the noise while disconnecting external amplifier and signal source. When the noise stops, the ground loop has been broken. Ground the device which caused the ground loop to same potential as the computer connected to the developer board, use optical digital audio interfaces ORX1 and OTX1 or isolate the signals galvanically.

3.3 Serial Port

If the operating system doesn't automatically install drivers when connecting the VSIDE UART cable to the USB port, you can download the driver for the cable from `http://www.vsdsp-forum.com/` - VSDSP tools - VSIDE.

If the USB port which VSIDE UART uses is changed, serial port number usually changes too. Creation of new port can happen. Microsoft Windows supports up to one hundred serial ports and beyond that its behavior is undefined. If this limit is reached, contact your system administrator to remove unused ports and reset the port number counter.

4 VSIDE

4.1 Installing VSIDE

Download the newest VSIDE from `http://www.vsdsp-forum.com/` - VSDSP tools - VSIDE and run the installer program vside_win32_v???.exe where ??? is the version number of VSIDE. The default installation path C:\Program Files (x86)\VSIDE has spaces and parenthesis which is just asking for a trouble. We strongly recommend using C:\VSIDE\ as installation target.

If everything went well, there should be VSIDE icon on your desktop. Also a menu entry can be found in the start menu.

Start VSIDE and study its help system. There is also VSIDE User's manual available in `http://www.vlsi.fi/en/support/download.html`. However it is obsolete in many parts.

In figure 1 is shown the most used buttons from toolbar in application development. Below is listed the functions of the buttons.

- Build solution - Compile changed files in solution and rebuild the binary
- Rebuild solution - Recompile all files in solution and rebuild the binary
- Clean solution - Remove build output files.
- Stop build - Stop running build
- Target drive - Target where to copy the ready binary.
- Run - Run stand-alone project on the developer board.

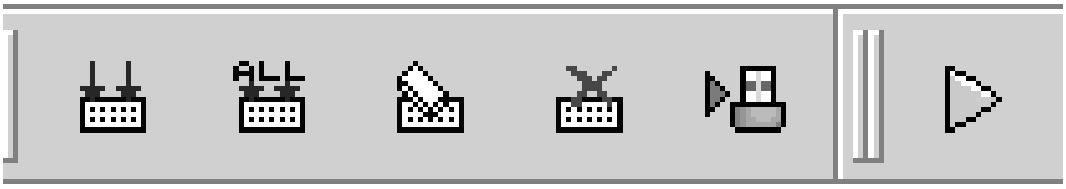

Figure 1: Application control buttons

4.2 Stand-Alone Application

Testing toolchain and stand-alone program is the next task. Start with File, New Project/solution. Use VS1005 solution as template solution and Hello world as template project, build and run it. It should open serial port and run hello world example. VSIDE will inform if communication doesn't succeed. When using serial port through VSIDE, terminal program has to be disconnected.

NOTE! If the VSOS Shell environment is active, it is not possible to connect to VS1005 through the serial port! If you are in the VSOS Shell environment, reset the VS1005 Developer Board while keeping BOOT SELECT (if VSOS is in external SPI FLASH) or S1 (if VSOS is in internal SPI FLASH) pushed. When you *don't* get the S:> prompt after reset, you can connect to the VS1005 Developer Board.

If program was compiled and ran successfully, in Standard Input/Output tab there should read "Hello, world!"

4.3 VSOS

With stand-alone project there is a lot of things to do. Much smarter way is to use VSOS and its services at least in case when end application should do something more complicated.

There might be newer release of VSOS `http://www.vsdsp-forum.com/` - VS1005 and VSOS Software - than bundled with VSIDE. Download kernel and extract the solution. Rebuild all and flash it to the board with prommer/flasher utility. Prommer/flasher gives different targets for kernel. In many cases external SPI flash is preferred target, so select VS1005G External SPI Flash Prommer. Then kernel installation is simple next - next - start procedure. It is strongly advised to read the dialogs and consider the target environment where VSOS will be running.

If you want to program the kernel into the internal SPI Flash of VS1005, you need to uncomment the following line near the beginning of vsos_vs1005g.c:

```
//#define USE_INTERNAL_FLASH
```

Then recompile the solution, and select VS1005G Internal Flash Prommer.

If flashing fails with **Error: could not load prommer application** error message and the developer board has VSOS installed with VSOS SHELL, developer board must be rebooted to such mode where VSOS SHELL isn't running. This can be achieved pressing S1 down and rebooting the board. About 2 MB USB disk should be present in that mode, or about 1 MB if the internal SPI Flash is used. Different starting modes are listed in table 1. The default mode is to boot to graphical interface. To make VSOS SHELL as default mode, press S1 and reset to access system drive. Edit the config.txt file and switch [0] to [2] and [2] to [0]. Save and safely remove the drive. This swaps the configuration number, so graphical user interface can be accessed by pressing S2 and restarting the developer board.

If developer board is brand new or something has broken the filesystem, the exported disk from developer board appears unformatted. In that case format it. All the data which was on it will be lost.

Just running kernel without any accessory programs doesn't help much. There is also a libraries and drivers package available. Download the newest version from `http://www.vsdsp-forum.com/` and extract it to somewhere where it is easily available. USB

Table 1: Different starting modes of the developer board

Pressed buttons	Config number	Starting mode
none	0	Graphical user interface
S1	1	USB mass storage of the system disk
S2	2	VSOS Shell. Use VSIDE UART and terminal emulator to connect.
S3	3	Not used.
S4	4	Classic player
S1 + S2	5	USB mass storage of the SD card
Boot select		Boot from internal SPI flash

cable on CN5 must be connected. Pressing S1 and reset on the developer board, boots VS1005 to the USB mass media mode. Removable drive should appear where applications and libraries should be copied.

In the root of the drive is config.txt which is used to setup the VSOS environment. After copying the VSOS libraries from directory named root of the extracted distribution package to root of VS1005 flash and possible configuration changes in config.txt, safely remove the drive and reset the developer board.

Developer board will print something similar to serial port when it starts

```
Hello.
VSOS 3.27 build Jun 03 2016 08:43:38
VLSI Solution Oy 2012-2016 - www.vlsi.fi

Starting the kernel..
Starting Devices...
External SPI Flash

Installed system devices:
S: SPI Flash c814, handled by FAT.
Load drivers, config 0...
Driver: SDSD... E'SD Card not found'
D: SD card in SD mode

Driver: AUODAC...
Driver: AUIADC...   Input 0x4040  Rate 48000
Driver: RUN... YBITCLR FC00,DFC00,D Y:0xfc00: 0x2000-13 -> 0x0
Driver: RUN... YBITCLR FC00,CFC00,C Y:0xfc00: 0x0-12 -> 0x0
Driver: UARTIN...
Driver: S:SHELL.AP3...
VSOS SHELL
S:>
```

5 VSOS SHELL

If not yet read, study VSOS UART shell document. It will give more comprehensive information how to use the shell. As a sneak peek some commands are listed below.

::	List available system devices
D:	go to SD card
S:	go to System disk
cd	Change directory
dir	List files in current directory, with parameter -a list only audio files
diskfree	Show amount of free space in disk
driver	Load, unload or query driver
help	Show short help of VSOS SHELL
playdir	Play all audio files from current directory
playfile	Play single audio file

As from the short list above can be seen, there are familiar concepts from DOS like drive letters and dir command. Output of help command shows familiar concepts of Linux terminal such as key bindings, command repetition and history.

VSOS SHELL gives ability to test and debug with different tools programs. Such accessories as liblist, liblist2 and frags can give information about system memory layout. Also if program references a NULL-pointer, a recovery environment is provided to investigate problem at hand. Memory and registers may be examined and modified with preg.

5.1 Using PuTTY to Connect to Your Develepor Board

PuTTY is terminal emulator software and SSH client. Its homepage is `http://www.chiark.greenend.org.uk/~sgtatham/putty/`. Install and run it.

Consult VS1005 VSOS Shell document how to set up correctly PuTTY. Basically the connection is 115200/8N1, no flow control.

Opening connection window is shown in figure 2. By default the terminal, keyboard and serial settings has to be edited as seen in VSOS Shell document. After settings are done a profile can be created by writing name for profile in saved sessions text input and clicking the save button. It makes connecting easier with doubleclicing the profile name opens the connection.

For a terminal emulator PuTTY has much use for mouse. One nice feature is mouse copy and paste. When text is selected, it is copied to clipboard. When right mouse

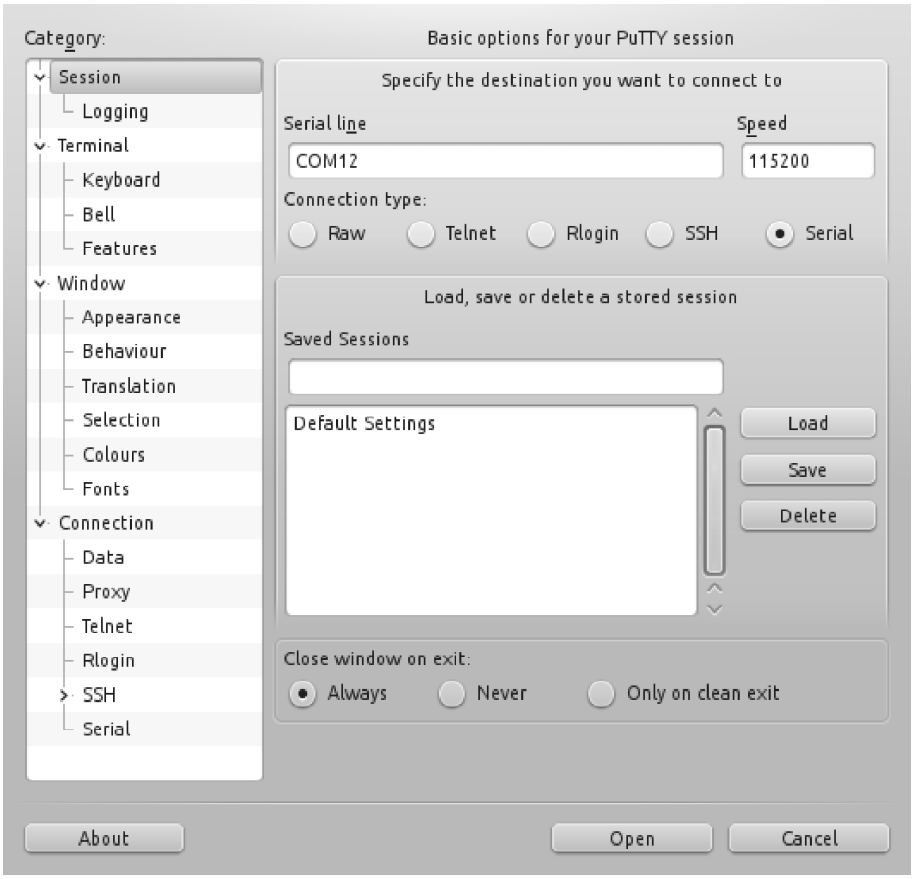

Figure 2: PuTTY session window when serial port is selected

button is clicked, text from clipboard is pasted. So beware of accidental pastes. Accessing preferences can be with right clicking the titlebar of running PuTTY session and selecting from drop down menu change settings.

Closing the connection is done by closing the program.

6 VSOS Application

VSOS is optimized operating system providing many UNIX-like interfaces. If something is to be read or write, there is high chance it will be a file and can be accessed through file pointer with fread(), fwrite() and controlled with ioctl(). VSOS is designed some object-oriented programming paradigms, such as inheritance and polymorphism. In normal application development these concepts doesn't restrict development, but make using the interfaces simpler. Dynamical loading of libraries is provided for preserving resources and is covered in section 7.3

By looking figure 4 notice how the processing is done in file MyMixerAudio. Also in section 7.4 is overriding default read() function to implement muting feature.

Writing application for VSOS with VSIDE is quite easy. Hello world application can be built straight from template and used as a starting point for your own application. Little bit more complicated application is done to demonstrate the features of VSOS and audio interface. This example works better if audio output can be monitored.

6.1 Starting with a Template

Start with File, New Project/solution, and select VS1005 VSOS3 application as solution template. Name the solution in meaningful way and click Next. Select form Project template VSOS3 Audio In Out. The project name names also the application which will be installed to system. The demo code in section 6.2 is named as twotone. So name the project and click Ok and there should be template project ready for further development.

Next stage is to make life little bit easier again. If developer board isn't in USB mass storage mode, reset it to that mode by holding S1 and pressing reset. 1 or 2 MB USB drive should become available.

From the toolbar click Target drive and the dialog should ask where to save the application. If target is DRIVE:\sys, VSOS SHELL can execute the application by its name. If the application is put to the root of the drive, VSOS SHELL needs the full path of the application, for example S:TWOTONE.AP3. Note the missing separator between S: and TWOTONE.AP3 Now that developer board is shown as mass storage, it is possible to copy the application straight to it.

If your developer board has some audio source connected to line in and output can be monitored, the compiling template will play the signal from input to output. See section 6.3 how to compile, install the application.

6.2 Demonstration Program

VSOS provides two FILE pointers for every application. For reading audio input stdaudioin is used and for writing audio to output stdaudioout is used.

Listing 1 has sorce code for application which writes two summed sine waves to output. Frequencies are selected to be purposely unpleasant. Also higher frequency is attenuated and lower frequency is enhanced as human hearing isn't linear.

Listing 1: main.c from twotone program

```
// File: main.c
#include <vo_stdio.h>
#include <stdlib.h>
#include <apploader.h> // Contains LoadLibrary() and DropLibrary()
#include <kernel.h> // Contains functions exported from kernel,
                    // like CoarseSine()
#define BUFSIZE 128 // Output buffer size
#define SIN_ATT 8   // Attenuation by shifting this many bits right
// Sample rate / buffer size = iteration count of while loop / second
#define WHILE_ITERATION 1875 // five seconds

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*BUFSIZE];
  s_int16 *p;
  u_int16 ph1 = 0, ph2 = 0, i, duration = WHILE_ITERATION;

  while (duration--) {

    // By default both input and output are 16-bit stereo at 48 kHz.
    p = myBuf;
    for(i = 0; i < BUFSIZE; i++){
      // Left channel
      // First Sine wave
      // CoarseSine() isn't HI-FI Sine wave. But for beeps
      // it is good enough and fast.
      *p = CoarseSine(ph1) >> (SIN_ATT - 1);
      // 65536 / 410 = 159 number of samples for one period.
      // Output frequency 48000/159 = 301.88Hz
      ph1+= 410;
      //Second sine wave
      // 65536 / 2730 = 24 number of samples for one period.
      // Output frequency 48000/24 = 2kHz
      p[0] += CoarseSine(ph2) >> (SIN_ATT + 1);
      ph2 += 2730;
      // Copy left channel to right channel.
      p[1] = p[0];
      // Advance p to next sample.
      p += 2;
    }
    // Write stereo samples from myBuf into stdaudioout.
    // By default, stdaudioout goes to line out.
    fwrite(myBuf, sizeof(s_int16), 2*BUFSIZE, stdaudioout);
  } /* while (duration--) */

  return EXIT_SUCCESS;
}
```

The program is rather straightforward. First some constants are defined. Then comes the main() function which has a buffer for writing and some iteration variables. While loop is executed WHILE_ITERATION times. The for loop generates the audio samples. CoarseSine() is used to make the samples. Its output is interpolated so it has some distortion. CoarseSine() takes u_int16 (unsigned 16-bit integer) as a phase parameter and returns a s_int16 (16-bit signed integer) value.

Sine function values are right shifted for attenuation. A terrible distortion would happen without right shifting in case where waves amplify each other and the result would overflow. VS_DSP^4 core provides hardware saturation features and the saturate.h header file has functions to control the overflow.

In the end of the for-loop left channel sample is copied as right channel sample. The last thing to do in the for-loop is to advance the buffer pointer.

After the for-loop has been executed, buffer is written to the stdaudioout. The while-loop starts again if the duration variable differs from 0.

Notice the structure of inner loop. Simple for loop is used with iteration starting, incrementing and ending. This way VS_DSP^4 core can use hardware to handle the loop. Normal indexing of buffer would work, but again buffer is accessed through pointer so that the last addition to next sample can be done on hardware with zero clock cycles. More information about good coding practices for efficient code can be found VS1005 VSOS Programmer's guide.

6.3 Running VSOS Application

When an application is ready to be tested on the developer board, it should be compiled and copied to the flash. If copying target was set as instructed in section 6.1 VSIDE can do this automatically.

Reset the Developer board to USB mass storage mode by holding S1 down and pressing reset. Click build solution and in the build log should be something similar.

```
copy loadable.ap3 d:\twotone.ap3 /y

Finished
```

Remember safely remove the disk and then reset the Developer board.

When VSIDE UART cable is connected as instructed in section 3.2 use terminal application of your choice to control VSOS SHELL. In VSOS SHELL run S:TWOTONE.AP3 and line out and phones should output the audio for a five seconds.

7 Introduction to Audio Subsystem

7.1 Audio Subsystem Overview

The VSOS audio subsystem is documented thoroughly in VS1005 VSOS Audio Subsystem document. This section covers some practical solutions for effective audio signal processing with VS1005. Dynamical loading is covered also in this section just before demonstrating library code.

In VSOS audio subsystem is simple for application. Reading from stdaudioin and outputting to stdaudioout are the basic tasks which can be done with fread and fwrite. Application doesn't have to consider what driver is really used or what else is on the signal path. The path is shown in figure 3. Other tasks, such as sample rate configuration or adjusting volume is done with ioctl() function.

Figure 3: Simple audio path in VSOS.

Audio source and output is done by loading library in to the memory. The name of audio library starts with AU and then comes the direction. I for input, O for output. If library starts with AUX, it can work with input or output. It doesn't have any connection with term "auxiliary port."

AUIADC is line input library to be used with analog inputs and AUOI2SMA is I^2S master output driver. Complete list and documentation for audio libraries can be found in the VS1005 VSOS Audio Subsystem document.

Audio driver libraries can be loaded from config.txt or interactive use there is VSOS command DRIVER. The order of loading audio libraries is important. First load the input driver, then pre-process driver which uses the input, output driver and post-process driver. Idea behind the order is input driver provides stdaudioin and pre-process driver uses it. Also output driver provides stdaudioout and post-process uses it. When all drivers are loaded, application can run. If it is required to unload the drivers, it is important to unload in reverse order which they were loaded. See the source code of init() and fini() functions in section 7.4 for reason of order restriction.

Implementing own filters and using own audio streams isn't complex. If only single audio path is required, using stdaudioin and stdaudioout as hard coded input and output can reduce the code to very compact implementation as seen in section 7.4.

In figure 4 is shown audio path for line in and audio file mixer. Arrow direction represents flow of audio data and used functions. First create stream MyAudioInput. Then load audiodec library and call CreateAudioCodec() with the file pointer and myAudioInput. MyAudioInput has write() function which consumes audio from stdaudioin and mixes it to stdaudioout. The main application has to handle user interface events and call proper ioctl() functions to control the signal processing. The propagation of ioctl() function is shown with dashed arrows. Considering reusability, this file type could be used even

with demo program shown in section 6.2. The output just wouldn't be stdaudioout but MyMixerAudio.

This is just an example design. Implementation is left as an exercise for the reader. Section 7.4 and source code of PlayFile, which can be found root file system and sources package distributed along VSOS, can be used as reference when implementing mixer file.

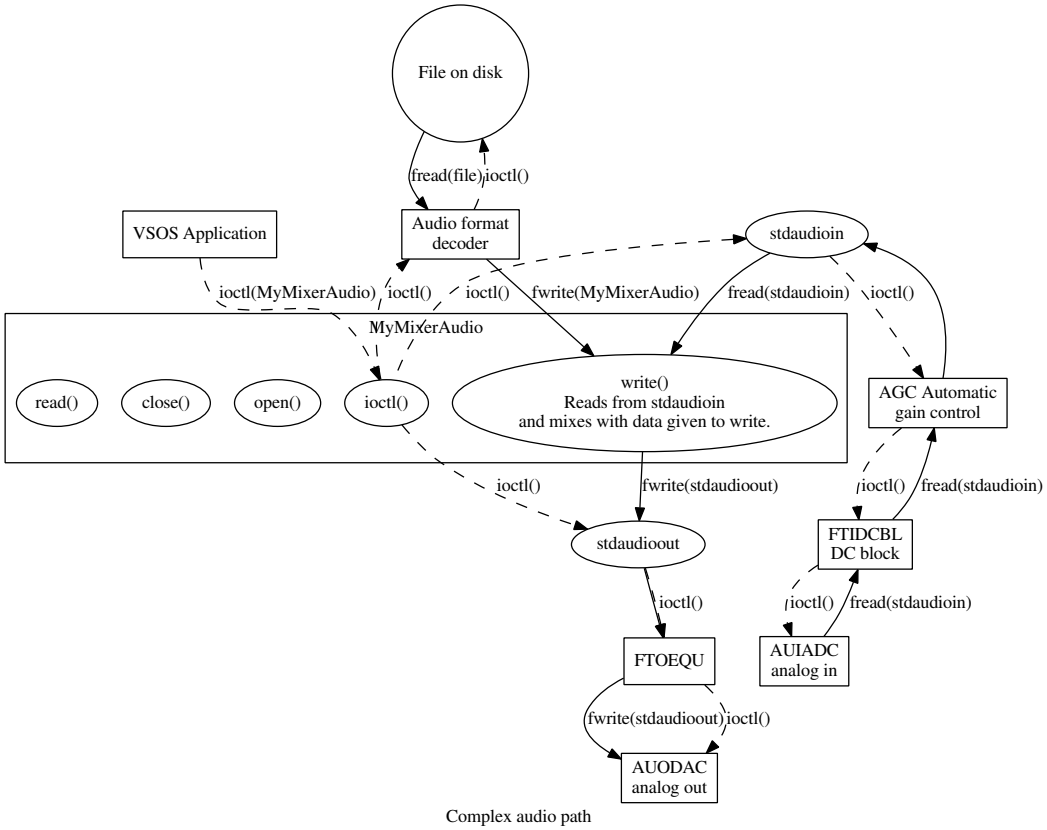

Figure 4: Audio path for mixing analog audio with audio file.

7.2 Post-Processing the Audio of the Demo Application

In section 6.2 implemented twotone.ap3 generates two sine waves, about 300Hz and 2kHz, which sound unpleasant together. In this section output is post-processed in VSOS SHELL by equalizer. The target signal path is shown in figure 5. The stdaudioin is there and available, but it isn't used so it isn't on signal path.

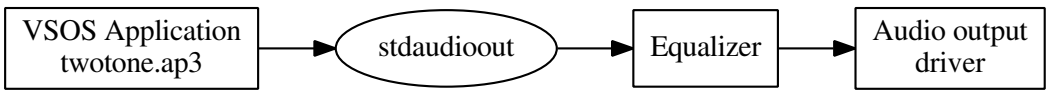

Figure 5: Equalizing audio in VSOS.

Information about audio path can be seen with commands “auinput” and “auouput” It is possible to load equalizer from VSOS SHELL interactively “driver +ftoequ” command.

After equalizer is loaded, it can be configured. To set the equalizer setequ command is used.

setequ 1 3 2000 −12 4

This setequ command sets filter 1 with channels 1 + 2 = 3 with center frequency being 2000 hertz gain to -12 dB. The Q factor specifies the bandwidth of the filter and can be calculated with $BW = \frac{f_c}{Q}$ which in this case gives 500Hz bandwidth. Running the twotone.ap3 before and after setting the equalizer, should make noticeable difference in played audio.

A steeper notch filter could be used for example removing incoming mains brum with similar command. The documentation of FTOEQU recommends using moderate equalization parameters to keep the audio quality.

Disabling equalizer can be done with command

setequ 1 0

which syntax means set equalizer filter 1 to no channels. Equalizer can be unloaded from the memory with “driver -ftoequ” command. Documentation of FTOEQU is delivered in RootAndLibrariesSource package which can be found from `http://www.vsdsp-forum.com/` - Software design - VS1005 and VSOS Software - Latest VSOS kernel.

Next is sample execution of the post-processed twotone.ap3. It begins by showing information about audio output. Methods of stdaudiout has everything is set to AUODAC, **AU**dio **O**utput **D**igital to **A**nalog **C**onverter.

Fil**T**er **O**utput **EQU**alizer is the description of the equalizer which is loaded with driver command.

The loading of equalizer can be seen in methods of stdaudioout. All but identify() method starts with FTOEQU::.

Before setting up the filters, the twotone.ap3 is run. 2kHz frequency is attenuated with setequ command and then twotone is run again to hear the difference.

Last operation is removing the equalizer driver and checking everything has returned as it was before post-processing demonstration.

```
S:>auoutput
stdaudioout:     0x1fea, AUODAC::audioFile=3139(0xc43)
  ->Identify():  0x36e6, AUODAC::Identify returns "AUODAC"
  ->op:          0x1ff1, AUODAC::audioFileOps=0(0x0)
    ->Ioctl():   0x35af, AUODAC::AudioIoctl
    ->Write():   0x369c, AUODAC::AudioWrite
Sample rate:     48000
Bits per sample: 16
```

```
Buffer size:     4096 16-bit words (2048 16-bit stereo samples)
Sample counter:  162035431
Underflows:      154589296
Volume:          +0.0 dB of maximum level
S:>driver +ftoequ
S:>auoutput
stdaudioout:     0x2551, FTOEQU::audioFile=2147(0x863)
  ->Identify():  0x36e6, AUODAC::Identify returns "AUODAC"
  ->op:          0x267b, FTOEQU::audioFileOps=0(0x0)
    ->Ioctl():   0x41d0, FTOEQU::AudioIoctl
    ->Write():   0x428e, FTOEQU::AudioWrite
Sample rate:     48000
Bits per sample: 16
Buffer size:     4096 16-bit words (2048 16-bit stereo samples)
Sample counter:  163412424
Underflows:      155964242
Volume:          +0.0 dB of maximum level
S:>s:twotone.ap3
S:>setequ 1 3 2000 -12 4
S:>s:twotone.ap3
S:>driver -ftoequ
S:>auoutput
stdaudioout:     0x1fea, AUODAC::audioFile=3139(0xc43)
  ->Identify():  0x36e6, AUODAC::Identify returns "AUODAC"
  ->op:          0x1ff1, AUODAC::audioFileOps=0(0x0)
    ->Ioctl():   0x35af, AUODAC::AudioIoctl
    ->Write():   0x369c, AUODAC::AudioWrite
Sample rate:     48000
Bits per sample: 16
Buffer size:     4096 16-bit words (2048 16-bit stereo samples)
Sample counter:  167872563
Underflows:      159944381
Volume:          +0.0 dB of maximum level
S:>
```

7.3 Dynamically Loaded Binaries in VSOS

In section 7.2 equalizer was loaded to memory and used to process the audio. VSOS doesn't differentiate between programs and libraries. Extensions, .AP3 and .DL3, doesn't make difference for the binary code. Main menu application searches for .AP3, not .DL3 files. On the other hand, VSOS SHELL commands and libraries use the .DL3 extension.

Libraries and programs are loaded to memory and unloaded when not needed anymore. Libraries have callable interface which is listed in table 2. Table 3 shows life time of the library with different loading methods.

Libraries can be loaded to memory with DRIVER +libraryname. First time when library

is loaded, init() and main() is run. If there is a requirement for another instance of library, DRIVER +libraryname will run only main(). DRIVER -libraryname executes fini() and unloads all instances of library. Thinking in object oriented programming init() is constructor and fini() is destructor.

There is also C-interface for dynamical loading. Header file apploader.h has the function prototypes. Noteworthy functions are RunLibraryFunction, LoadLibrary and DropLibrary.

Function int RunLibraryFunction(const char* filename, u_int16 entry, int i) might be the most common function to use. The filename parameter is name of the library file, entry is often constant ENTRY_MAIN and int i is parameter for the function. RunLibraryFunction loads library with LoadLibrary() function from file to the memory, executes the function, unloads the library with DropLibrary() and returns the return value of the called function.

Table 2: Library interface functions

Function	Event of execution
init()	When library is loaded to memory
main()	In the start of every instance of the library
fini()	When library is unloaded from memory

Table 3: Library life time in memory

Loading method	Loaded to memory	Unloaded from memory
“lib.dl3” in config.txt	On reset	Never or with “driver -lib”
“run lib.dl3” in config.txt	On reset	On exit of main()
“lib.dl3” in shell	On invocation	On exit of main()
“driver +lib” in shell	On invocation	Never or with “driver -lib”

7.4 Creating Your Own Audio Filter

Sometimes it is required to add some signal processing on the audio signal path. It can be implemented inside the application, but for easier testing and reusability it is better implement in library which pre-process stdaudioin or post-process stdaudioout.

Next demonstration is to design simple cough button for pre-processing input. When button S1 is pressed, audio input is muted. Then it is developed further in section 7.5 to a crude noise gate effect which mutes the signal if it stays under selected value. The desired audio path is shown in figure 6. The name of cough button comes from the broadcasting domain where people behind the microphones have button to mute the microphone temporarily for example while coughing.

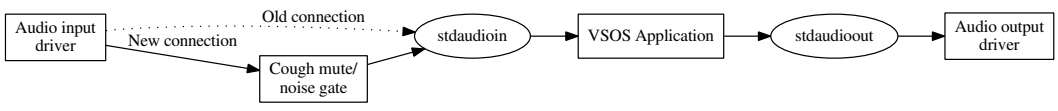

Figure 6: Audio path of the cough mute and noise gate filter for stdaudioin.

Filter uses SIMPLE_FILE data structure which inherits FILE pointer's interface. In figure 7 is how different fields are populated. To implement the filter, only init() and fini() are required for library. The SIMPLE_FILE data structure has identify() overloaded and ioctl(), read() in operations field. The noisegate file is populated in init() function and inOrg is set to point to stdaudioin. After that stdaudioin is overwritten to point noisegate. In fini() function the stdaudioin is set back to inOrg and noisegate is gone from the audio signal path.

NoisegateIdentify() function just returns text to identify which driver implements the file type. This information is used by auinput command to display it for a user.

Ioctl() function is used to change parameters of audio. Best practice with ioctl commands is to check if own state change is needed and forward the command and return whatever th last in chain returns. This can be seen in figure 4.

In NoisegateRead() function is shown one of the most important concepts to keep in mind with all VS_DSP[4]systems. The destination index makes possible to shift the data by 8-bit resolution and is a way to read 8-bit values to 16-bit variables "fread(fp, &var, 1, 1);" will do that. If destinaitionIndex is odd, first 8-bit value is stored in MSB. **There is no 8-bit data type.**

NoisegateRead() function is designed to handle odd index of read function. The solution is to buffer the samples and get nice alignment for 16-bit data type. However it is possible to read partial samples, which should be avoided.

There is no need to define the main() function. It is left in the source code as it will be populated in section 7.5.

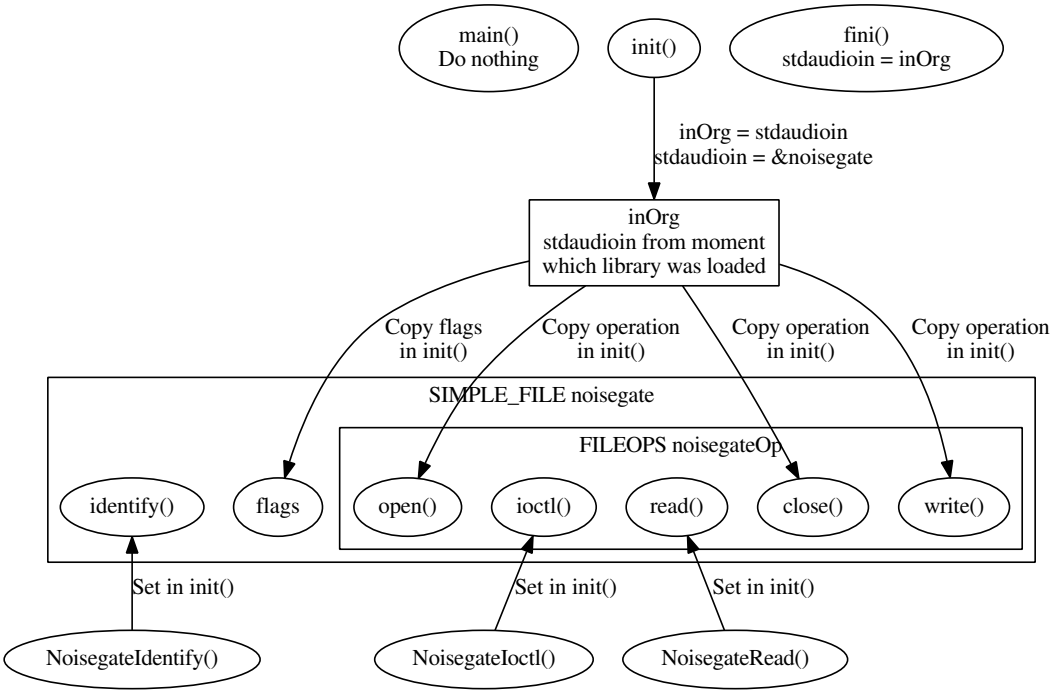

Figure 7: Data structure of the cough mute and noise gate filter for stdaudioin.

Listing 2: Noise gate project main.c file

```
/* For free support for VSIDE, please visit www.vsdsp-forum.com */

// From solution noisegate file : main.c

// Starting point template for creating VSOS3 libraries and device drivers.
// This will create a <projectname>.DL3 file, which you can copy to
// your VS1005 Developer Board's system disk's SYS subdirectory.

// If init (), main() or fini () are not needed, remove them from the solution.
// There's no need to have any unneeded functions in the library.

#include <vo_stdio.h>
#include <volink.h>       // Linker directives like DLLENTRY
#include <apploader.h> // RunLibraryFunction etc
#include <string.h>
#include <stdlib.h>
#include <vo_gpio.h>
#include <aucommon.h>


FILE *inOrg = NULL;
u_int16 wordsPerSample = 1;
extern FILEOPS noisegateOp;
extern SIMPLE_FILE noisegate;

// This function is called when the library is loaded.
// If CONFIG.TXT has several instance of the same driver,
// init () is called only once.
void init (void) {
  // Copy not implemented functions from stdaudioin.
  inOrg = stdaudioin;
  noisegate.flags = stdaudioin->flags;
  noisegateOp.Open = stdaudioin->op->Open;
  noisegateOp.Close = stdaudioin->op->Close;
  noisegateOp.Write = stdaudioin->op->Write;
  // Set stdaudioin pointing to noisegate.
  stdaudioin = (VO_FILE *)&noisegate;
}

char* NoisegateIdentify(register __i0 void *self,
                        char *buf,
                        u_int16 bufsize) {
  return "Noise Gate";
}

IOCTL_RESULT NoisegateIoctl (register __i0 VO_FILE *self,
                             s_int16 request,
                             IOCTL_ARGUMENT arg) {
  IOCTL_RESULT ret;

  //Pass IOCTL request to the original stdaudioin
  ret = inOrg->op->Ioctl(inOrg, request, arg);
  // If call touched bits, read them back
```

```
  // to get current values.
  if (request == IOCTL_AUDIO_SET_RATE_AND_BITS ||
      request == IOCTL_AUDIO_SET_BITS      ||
      request == IOCTL_AUDIO_SET_IRATE) {
    wordsPerSample = inOrg->op->Ioctl(inOrg, IOCTL_AUDIO_GET_BITS, NULL)>>4;
  }
  // Return original value.
  return ret;
}

u_int16 NoisegateRead (register __i0 VO_FILE *self,
                       void *buf,
                       u_int16 destinationIndex,
                       u_int16 bytes) {
  /* If this read function was something else than audio,
     read and mute should be implemented in a way which
     could handle the odd destinationIndex and any number
     of bytes.

     In case of audio, the read must be stereo pair of samples.
     16-bit: bytes %4 == 0 32-bit bytes % 8 == 0
  */
#if 0
  static s_int16 tmpBuf[128];
  u_int16 b = bytes;
  // This way odd destinationIndex won't break muting.
  while (bytes) {
    if (b > 256) {
      b = 256;
    }
    inOrg->op->Read(inOrg, tmpBuf, 0, b);

    if (GpioReadPin(0x00)) { // Button S1 pressed
      memset(tmpBuf, 0, sizeof(tmpBuf));
    }
    MemCopyPackedBigEndian(buf, destinationIndex, (u_int16*)tmpBuf, 0, b);
    destinationIndex += b;
    bytes -= b;
  }
#else
  inOrg->op->Read(inOrg, buf, destinationIndex, bytes);
  if (GpioReadPin(0x00)) { // Button S1 pressed
    memset(buf + destinationIndex, 0, (bytes>>1));
  }
#endif
}

// Startup code for each instance of the library
// If CONFIG.TXT has several instance of the same driver,
// this is called for each line.
ioresult main(char *parameters) {
}

// Library finalization code.
```

```
// This is called when the library is dropped from memory,
// (reference count drops to zero due to a call to DropLibrary)
void fini (void) {
  // Set stdaudioin back to its original state.
  stdaudioin = inOrg;
}

//temporarily switch off compile warning about different objects
#pragma msg 30 off
FILEOPS noisegateOp = {
  CommonOkResultFunction, //AudioOpen,
  CommonOkResultFunction, //AudioClose,
  NoisegateIoctl,
  NoisegateRead,//Not real AudioRead
  CommonOkResultFunction, //AudioWrite
};
#pragma msg 30 on


SIMPLE_FILE noisegate = {
  0, // flags: not present
  NoisegateIdentify, // Identify () function
  &noisegateOp, // Fileoperations from above.
};
```

Install compiled binary to SYS/ folder. See section 6.1 how to set installation folder. In VSOS SHELL use DRIVER +noisegate to load the library. Provide some input signal to line-in or select microphone with “auinput mic1 mic2” command as input source. Command “loopback” should be used to read from stdaudioin and write to stdaudioout. When button S1 is pressed, audio should be muted.

7.5 Modifying Cough Button to Noise Gate

This section is rather heavy on source code listings. A little bit more complex example is selected to show some signal processing and utilities of VSOS.

The filter from section 7.4 is just a cough button. Next features are added to make it real noise gate. In figure 7 main() function has description of do nothing. Also table 2 gives important information. When library is loaded, init() and main() functions are called. When library is unloaded fini() is called. Every other instance is done by calling main() function. These features are used as advantage. If main() function takes parameters which set or query operating values and without parameters sets to them default value, the usage is shown in table 4.

Table 4: Usage of noisegate

Command	Status	Action
driver +noisegate	Not loaded	Loads noisegate to memory and sets gate threshold to default value.
driver +noisegate -p	Not loaded	Loads noisegate to memory and prints current (default) values of operating parameters.
driver +noisegate -t <value>	Not loaded	Loads noisegate to memory and sets threshold value to <value>.
driver -noisegate	Not loaded	**No operation.**
noisegate	Not loaded	Loads to memory and unloads immediately. **Effectively no operation.**
noisegate -p	Not loaded	Loads to memory prints default gate threshold and unloads immediately. **Effectively no operation.**
noisegate -t <value>	Not loaded	Loads to memory sets gate threshold value to <value> and unloads immediately. **Effectively no operation.**
driver +noisegate	Loaded	Sets gate threshold to default value. **Extra overhead by driver.**
driver +noisegate -p	Loaded	Prints current gate threshold value. **Extra overhead by driver.**
driver +noisegate -t <value>	Loaded	Sets gate threshold value to <value>. **Extra overhead by driver.**
driver -noisegate	Loaded	Unloads from memory.
noisegate	Loaded	Sets gate threshold to default value.
noisegate -p	Loaded	Prints current gate threshold value.
noisegate -t <value>	Loaded	Sets gate threshold value to <value>.
noisegate -a <value>		Sets attack time to <value> μs
noisegate -w <value>		Sets hold (Wait) time to <value> ms
noisegate -r <value>		Sets release time to <value> ms
noisegate -h		Prints usage help

Table 4 gives all possible combinations for setting and getting threshold values. Running

noisegate without loading it to memory first is useless. However using driver +noisegate after noisegate is in the memory isn't smart either. It increases reference count of library and has overhead for the main() function. This lefts top three and bottom four commands to be usable. In section 7.3 covers the right use of driver command. Table 4 lists also other setup and help parameters.

The input signal is processed to behave as shown in figure 8. When the input signal goes above threshold value the output starts fading in the signal. Fade in time is attack time. After fade in, signal is unmodified while staying above the threshold. When signal goes below threshold, hold time is activated. When hold time has run out, the signal is faded out. Time which takes to fade out the signal is release time. If signal goes above threshold during hold or release time, the gate starts opening again. However opening gate won't start closing if signal goes below threshold. Attack times are usually few μs long not to make terrible distortions while hold and release times are often 5-200 ms.

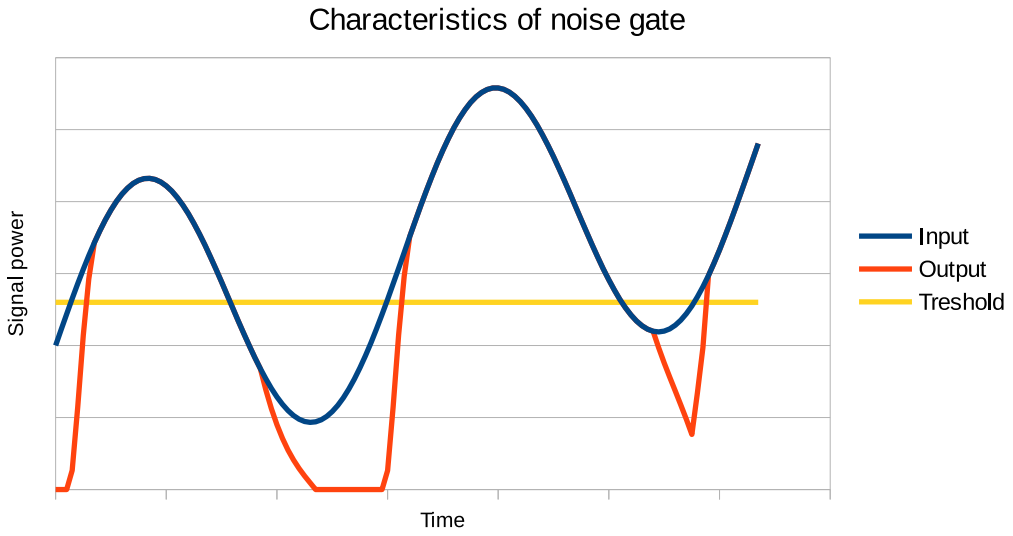

Figure 8: Amplitude response of the noise gate

To add the functionality, first the signal processing part is added and then the main() function is populated. After that the NoisegateRead() is modified to run the processing. To add a common header file, select File - New - New source file. Copy contents of listing 3.

Listing 3: Noise gate project noisegate.h file

```
#ifndef NOISEGATE_H
#define NOISEGATE_H
#include <vstypes.h>
// Audio stream dependent variable
extern u_int16 wordsPerSample;

// Setup variables

extern u_int16 noisegateDBThreshold;
extern u_int16 noisegateAttackUs;
extern u_int16 noisegateHoldMs;
extern u_int16 noisegateReleaseMs;

// Setup function
void NoisegateSetParameters(register u_int16 thresholdDb,
                            register u_int16 attackUs,
                            register u_int16 holdMs,
                            register u_int16 releaseMs);

// Signal processing function.
void NoisegateProcess(register s_int16 *p, register u_int16 samples);

// Helper functions to convert db to linear
auto u_int16 DBToLin(u_int16 n);
#endif
```

Save the file as noisegate.h. If the file doesn't show up under the Header files in the Solution browser, right click the Header files group and select add existing item. Then select the noisegate.h and "Open".

In main.c file add one include line after include all other include directives.

```
#include "noisegate.h"
```

Now the functions can find the variables which are used to control the noisegate. The gate is implemented in file noisegate.c. Create a new file and save it as noisegate.c and add it to the source files. Source code is provided in listing 4.

Listing 4: Noise gate project noisegate.c file

```
#include <string.h> //memset
#include <vsos.h> //ioctl
#include <aucommon.h> // IOCTL audio defines
#include <vo_stdio.h>
#include "noisegate.h"

// Setup values from main()
u_int16 noisegateDBThreshold = 0;
u_int16 noisegateAttackUs = 100;
u_int16 noisegateHoldMs = 100;
u_int16 noisegateReleaseMs = 20;
// Processing values.
u_int16 noisegateAttackSpeed;
u_int16 noisegateReleaseSpeed;
u_int32 noisegateThreshold; //Mean square value
u_int32 noisegateHold;
// Contextual variables
u_int16 noisegateMultiplier; //Fader multiplier.

#define NG_FULL_OPEN 65535
#define NG_FULL_CLOSED 0
/* Fades only 16 MSb's.
   If samples are 32-bit, this leaves 16 LSb's untouched.
   While not entirely correct from a signal processing standpoint,
   pseudo-random noise in the 16 LSb's during fading are insignificant. */
void FadeOut(register s_int16 *d, register u_int16 samples) {
  u_int16 mult = noisegateMultiplier;
  s_int16 i;
  d += wordsPerSample-1;
  for (i=0; i<samples; i++) {
    *d = (s_int16)(((s_int32)(*d) * (s_int32)mult) >> 16);
    d += wordsPerSample;
    mult -= noisegateReleaseSpeed;
    if (mult > noisegateMultiplier){
      // Underflow. Zero one sample too early and
      // the rest and then return.
      noisegateMultiplier = NG_FULL_CLOSED;
      memset(d, 0, (samples - i) * wordsPerSample);
      return;
    }
    noisegateMultiplier = mult;
  }
}

/* Fades only 16 MSb's.
   If samples are 32-bit, this leaves 16 LSb's untouched.
   While not entirely correct from a signal processing standpoint,
   pseudo-random noise in the 16 LSb's during fading are insignificant. */
void FadeIn(register s_int16 *d, register u_int16 samples) {
  s_int16 i;
  d += wordsPerSample-1;
  for (i=0; i<samples; i++) {
    *d = (s_int16)(((s_int32)(*d) * (s_int32)noisegateMultiplier) >> 16);
```

```
      d += wordsPerSample;
      if (noisegateMultiplier >= NG_FULL_OPEN - noisegateAttackSpeed){
        noisegateMultiplier = NG_FULL_OPEN;
        return;
      }
      noisegateMultiplier += noisegateAttackSpeed;
    }
  }
  /* Calculates only 16MSb's
     If samples are 32-bit, this doesn't count 16 LSb's
     Those would only provide useless accuracy and make
     calculation harder. Returns mean square value. */
  u_int32 CalcPower(const s_int16 *p, u_int16 samples) {
    u_int32 sumLo=0, sumHi=0, oldSumLo=0;
    s_int16 i;
    p += wordsPerSample -1;
    for (i=0; i<samples; i++) {
      sumLo += (s_int32)*p * (s_int32)*p;
      p += wordsPerSample;
      if (sumLo < oldSumLo) {
        sumHi++;
      }
      oldSumLo = sumLo;
    }
    // When bigger scale (not resolution) is needed,
    // use floating point numbers and beware of rounding errors!
    return (u_int32)((sumLo + sumHi*4294967296.0)/samples);
  }

  #define GATE_CLOSED 0
  #define GATE_OPENING 1
  #define GATE_OPEN 2
  #define GATE_CLOSING 3

  /* When NoisegateProcess gets signal above gateThreshold,
     opening is started with FadeIn(). After gate is open,
     level is monitored and if it goes below threshold,
     hold counter is started. When counter expires FadeOut()
     is called. If signal goes above threshold level, closing
     is stopped and opening is started again.*/
  void NoisegateProcess(register s_int16 *p, register u_int16 samples){
    static u_int16 state = GATE_CLOSED;
    static u_int32 holdCounter = 0;
    u_int32 power;
    // Everything is above zero so noise gate disabled or no samples.
    if (!noisegateThreshold||!samples){
      return;
    }
    power = CalcPower(p,samples);

    switch(state){
    case GATE_CLOSED:
      if (power < noisegateThreshold){
        memset(p, 0, samples * wordsPerSample);
```

```
        break;
      }
      state = GATE_OPENING;
      // Intentional fall-through to opening.
    case GATE_OPENING:
      FadeIn(p,samples);
      if (noisegateMultiplier == NG_FULL_OPEN){
        state = GATE_OPEN;
      }
      break;
    case GATE_OPEN:
      if ( power > noisegateThreshold){
        break;
      }
      state = GATE_CLOSING;
      holdCounter = noisegateHold;
      // Intentional fall-through to closing
    case GATE_CLOSING:
      if (power > noisegateThreshold){
        state = GATE_OPENING;
        // Fade in now and let next round put the gate open state.
        FadeIn(p,samples);
        break;
      }
      // Wait for hold time
      if (holdCounter > samples){
        holdCounter -= samples;
        break;
      }
      // Fade out
      FadeOut(p + (u_int16)holdCounter*wordsPerSample,
              samples - (u_int16)holdCounter);
      holdCounter = 0;

      if (noisegateMultiplier == NG_FULL_CLOSED){
        state = GATE_CLOSED;
      }
      break;
    default:
      // This should never run.
      state = GATE_CLOSED;
    }
}

void NoisegateSetParameters(register u_int16 thresholdDb,
                            register u_int16 attackUs,
                            register u_int16 holdMs,
                            register u_int16 releaseMs){
  u_int32 tmp, sampleRate;
  // Comparison is done with RMS values of the signal,
  // without the square root.
  noisegateThreshold = DBToLin(thresholdDb);
  noisegateThreshold = noisegateThreshold * noisegateThreshold;
  //Get the sample rate.
```

```
  if ( ioctl (stdaudioin, IOCTL_AUDIO_GET_IRATE, (char *)(&sampleRate))) {
    printf ("Couldn't get samplerate\n");
    return;
  }
  // Get bits.
  wordsPerSample = ioctl(stdaudioin, IOCTL_AUDIO_GET_BITS, NULL)>>4;
  //Calculate speeds.
  tmp = (sampleRate * attackUs / 1000000) + 1;
  // Not exact, but doesn't divide by zero in any case.
  noisegateAttackSpeed = (u_int16)(65535 / tmp);
  noisegateHold = sampleRate/1000 * holdMs;
  tmp = sampleRate / 1000 * releaseMs + 1;
  tmp &= 0xffff;
  noisegateReleaseSpeed = (u_int16)(65535 / tmp);
  noisegateDBThreshold = thresholdDb;
  noisegateAttackUs = attackUs;
  noisegateHoldMs = holdMs;
  noisegateReleaseMs = releaseMs;
}
```

The noisegate.c file has five functions. Setup function, fade in, fade out, calculate the mean square value of the buffer and processing function which uses three previous functions. Fade in and out multiply samples and increment or decrement the multiplier. However multiplying is done with 16 MSb's expanding to 32 bits and collapsing back to 16 bit.

The power calculation function shows 64-bit counter. It returns 32-bit mean square value.

The processing function is about control logic for hold delay and fade in and out functions. It is possible to write it more compact way, however verbose switch-clause was chosen for readability.

The setup function is needed for keeping the timings constant and set right sample width. Consider how the system is controlled. A decibel threshold level is given from main() function to setup function. The level is converted to RMS value. However the square root would be taken from every buffer. The RMS threshold value is squared and everything works nicely and there is no need to run heavy calculation of square root. The multipliers for fade in and out are also calculated. Information about sample rate is required for that.

The dbtolin.c file must be added to the project. These functions are fast and can be used in other projects where conversion from decibels to linear values or other way is required. The LinToDB() isn't even used in this noisegate, but it is provided for completeness. The source code is in listing 5.

Listing 5: Noise gate project dbtolin.c file

```
#include <vstypes.h>
const u_int16 linToDBTab[5] = {36781, 41285, 46341, 52016, 58386};

/*
  Converts a linear 16-bit value between 0..65535 to decibels.
    Reference level: 32768 = 96dB (32767 = 95dB).
  Bugs:
    - For the input of 0, 0 dB is returned, because minus infinity cannot
      be represented with integers.
    - Assumes a ratio of 2 is 6 dB, when it actually is approx. 6.02 dB.
*/
auto u_int16 LinToDB(u_int16 n) {
  int res = 96, i;

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

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

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

  return res;
}

const u_int16 DBToLinTab[6] = {
#if 0
  /* Correct rounding, so perfect complement to LinToDB() */
  34716, 38968, 43740, 49097, 55109, 61858
#else
  /* Gives nicer powers of two but does not exactly complement LinToDB() */
  32768, 36781, 41285, 46341, 52016, 58386
#endif
};

auto u_int16 DBToLin(u_int16 n) {
  u_int16 res = 1;
  if (n < 6) {
    return 0;
  }
  while ((n-=6) >= 6) {
    res <<= 1;
  }
  res = (u_int16)(((u_int32)res * DBToLinTab[n] + (1<<14)) >> 15);
  return res;
}
```

To achieve described functionality described in table 4, main() function is populated. Contents of main() in listing 6.

Listing 6: Noise gate project main.c file main() function

```
ioresult main(char *parameters) {
  int nParam, i;
  u_int16 *nextValue = NULL;
  char *p = parameters;
  ioresult ret = S_OK;
  nParam = RunLibraryFunction("ParamSpl", ENTRY_MAIN, (u_int16)parameters);
  for(i = 0; i < nParam; i++){
    if (!strcmp(p, "-h")){
      printf ("Usage: noisegate [-p|-t lim|-a usec|-w msec|-r msec|-h]\n"
              "-p\tPrint the threshold level, attack, hold and release times.\n"
              "-t\tSet threshold limit dB.\n"
              "lim\tThreshold limit in decibels\n"
              "-a\tSet attack time.\n"
              "usec\tMicrosecond (10^-6 s)"
              "-w\tSet hold time.\n"
              "msec\tMillisecond (10^-3 s)\n"
              "-r\tSet release time.\n"
              "-h\tShow this help\n"
              " If no parameters are given, default threshold value is set.\n"
              "Timing isn't changed\n");
      goto finally ;
    } else if (!strcmp(p, "-p")){
      printf ("Noisegate threshold: %02u dB, att: %03u us, hold: %03u ms, "
              "rel : %03u ms\n", noisegateDBThreshold, noisegateAttackUs,
              noisegateHoldMs,noisegateReleaseMs);
      goto finally ;
    } else if (!strcmp(p, "-t")){
      nextValue = &noisegateDBThreshold;
    } else if (!strcmp(p, "-a")){
      nextValue = &noisegateAttackUs;
    } else if (!strcmp(p, "-w")){
      nextValue = &noisegateHoldMs;
    } else if (!strcmp(p, "-r")){
      nextValue = &noisegateReleaseMs;
    } else if (nextValue){
      char *tmp;
      s_int32 val = strtol (p, &tmp, 0);
      if (*tmp == '\0'){
        if (nextValue == &noisegateDBThreshold){
          if (val < 0){
            val = 96 + val;
          }
          if (val > 96){
            goto bad_value;
          }
          *nextValue = (u_int16)val;
          nextValue = NULL;
        } else {
          if ( val >= 0 && val < 65536){
            *nextValue = (u_int16)val;
```

```
            nextValue = NULL;
          } else {
            goto bad_value;
          }
        }
      } else {
        goto bad_value;
      }
    } else {
      printf ("E: Unknown parameter \"%s\"", p);
      ret = S_ERROR;
      goto finally ;
    }
    p += strlen(p) +1;
  }
  if (!nParam){
    // Default threshold (noisegate disabled)
    noisegateDBThreshold = 0;
  }
  NoisegateSetParameters(noisegateDBThreshold,
                         noisegateAttackUs,
                         noisegateHoldMs,
                         noisegateReleaseMs);
finally :
  return ret;
bad_value:
  printf ("E: Bad value \"%s\"", p);
  ret = S_ERROR;
  return ret;
}
```

Notice how the parameters are handled. ParamSpl library is loaded and parameters are tokenized. After that, library is dropped. Next the parameters are parsed and the setup function is called.

To trigger the processing, NoisegateRead() is modified. An else branch is added where the signal processing is done. The modified part of code is in listing 7.

Listing 7: Noise gate project main.c file part of NoisegateRead() function

```
if (GpioReadPin(0x00)) { // Button S1 pressed
  memset(buf + destinationIndex, 0, (bytes>>1));
} else {
  NoisegateProcess(buf + destinationIndex, bytes/(2*wordsPerSample));
}
```

When implementing Read() or Write() functions, always think about generality. Will it work even destinationIndexes? How about odd index? Will it work when bytes is odd? The NoisegateRead() function buffers the read so destinationIndex has no effect. The bytes parameter is little bit more trickier. The calling of NoisegateProcess() function rounds the byte count down to sample level, so the last sample part is untouched in tmpBuf and it is copied to destination buffer. Next time when NoisegateRead() is called,

the input stream is off by a half sample. The audio will get corrupted on every second read. The workaround is simple. Remember call NoisegateRead() function with bytes count which results full samples.

Last modification is with NoisegateIoclt() function. For keeping the gate opening and closing times constant after sample rate is changed the NoisgateSetParameters() has to be called. The new code for NoisegateIoctl() is shown in listing 8.

Listing 8: Noise gate project main.c file NoisegateIoctl() function

```
IOCTL_RESULT NoisegateIoctl (register __i0 VO_FILE *self,
                             s_int16 request,
                             IOCTL_ARGUMENT arg) {
  IOCTL_RESULT ret;

  //Pass IOCTL request to the original stdaudioin
  ret = inOrg->op->Ioctl(inOrg, request, arg);
  // If call touched bits or sample rate, read them back
  // to get current values.
  if (request == IOCTL_AUDIO_SET_RATE_AND_BITS ||
      request == IOCTL_AUDIO_SET_BITS      ||
      request == IOCTL_AUDIO_SET_IRATE) {
    NoisegateSetParameters(noisegateDBThreshold,
                           noisegateAttackUs,
                           noisegateHoldMs,
                           noisegateReleaseMs);
  }
  // Return original value.
  return ret;
}
```

After these modifications, the project should compile and install nicely to S:SYS. Use “driver +ftidcbl” to drop the DC component from input signal. With command “driver +noisegate -t -50” noise gate is set to close below -50dB of the maximum of the signal. Decibels can be negative, then the reference is top of the signal, positive and the reference is zero level or zero when noise gate is disabled. To listen how noise gate works, connect earphones or line out to monitor the output and command “loopback” CTRL-C key combination ends the loopback command. The level is rather high for testing purposes. Low volume signal doesn’t open the gate and also pressing S1 button mutes the audio.

8 Linux-Specific Notes

Officially VSIDE runs on Microsoft Windows. However Wine is able to run VSIDE and there are many “Works for me” reports. This section contains information about best practices getting VSIDE running on Linux. Also some background information is given on various aspects of Linux and VSIDE. Installation and operating procedures are tested to work with Kubuntu 14.04, Wine 1.6.2 and VSIDE 2.39.

Main communication way between developer board and VSIDE is serial port which has to be available for VSIDE. Developer board can be connected also as mass media when installing libraries and applications for VSOS. Summarized process is shown below.

1. Prepare the serial port
2. Install VSIDE with Wine
3. Run application on board
4. Add application to S:\SYS\

8.1 Serial Port as a Device

When VSIDE UART is plugged in, it can be seen on Linux. On insertion kernel sends message to dmesg and which can be read with command dmesg.

```
[3995912.617405] usb 1-4: new full-speed USB device number 5 using xhci_hcd
[3995912.745972] usb 1-4: New USB device found, idVendor=067b, idProduct=2303
[3995912.745974] usb 1-4: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[3995912.745975] usb 1-4: Product: USB-Serial Controller
[3995912.745976] usb 1-4: Manufacturer: Prolific Technology Inc.
[3995912.823328] usbcore: registered new interface driver usbserial
[3995912.823340] usbcore: registered new interface driver usbserial_generic
[3995912.823350] usbserial: USB Serial support registered for generic
[3995912.829493] usbcore: registered new interface driver pl2303
[3995912.829521] usbserial: USB Serial support registered for pl2303
[3995912.829547] pl2303 1-4:1.0: pl2303 converter detected
[3995912.830440] usb 1-4: pl2303 converter now attached to ttyUSB0
```

Also command lsusb should show some similar line

```
Bus 001 Device 005: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port
```

Depending how your system is configured, serial port device can be /dev/ttyUSB0 or /dev/ttyACM0. Numbers can be different. In dmesg output there is line starting with

pl2303 which indicates Profilic driver has been loaded. Next line in the dmesg output tells which device it is. In this case it is ttyUSB0.

If device file for serial port isn't in /dev tree, make sure pl2303 kernel module is loaded and the USB port is working. Device won't appear if kernel has been upgraded but system hasn't been rebooted. In that case reboot solves the problem.

8.2 Serial Port as a File

Now when serial port has been loaded and device can be seen as a file, it is good time to check its permissions.

```
$ ls -l /dev/ttyUSB0
crw-rw---- 1 root dialout 188,  0 May 30 13:35 /dev/ttyUSB0

$ id
.... groups=20(dialout) ....
```

In this case user root, members of group dialout can read and write to this serial port.

If user wasn't in dialout group, there are two ways to handle the problem. One is to modify the file by writing such udev rule that user gets access to it by setting such group where the user is member, making user owner of the serial port or granting everybody read and write access. Another way to handle that kind of situation is to add user to dialout group. User must log off and log in for group modification to become effective. Consult documentation of your distribution how to make this configuration. Conventions differ from distribution to an another.

To be sure serial port works, open a connection with serial terminal application, such as Putty. Communication parameters are 115200 bps, 8 data bits, no parity, one stop bit and no control flow. Developer board may answer something or not. Important is that it is possible to use the port.

8.3 Wine and VSIDE

Microsoft Windows programs don't normally run on Linux. However it is possible to install Wine which adds some compatibility for Windows programs.

Consult documentation of your distribution how to install Wine. Custom settings can be done with winecfg program. Wine has quite good documentation which can be accessed from `https://www.winehq.org/`

By default Wine doesn't provide any serial ports for applications. Granting access for VSIDE to use the serial port is rather simple but important task.

With Command

cd ~/.wine/dosdevices && ln −s /dev/ttyUSB0 com1 && cd

1. Current directory is changed to user's home directory/.wine/dosdevices
2. If the directory change was successful then symbolic link com1 is created to point /dev/ttyUSB0.
3. If creation of symbolic link was successful go back to user's home directory

If directory ~/.wine doesn't exist run

wine winemine

and play a game of minesweeper. After that wine should have created the ~/.wine directory and it is possible to add the serial port.

VSIDE installation differs from Microsoft Windows in two aspects. Installation is started with wine and using installation target c:\VSIDE\ becomes more important.

To install VSIDE, download newest VSIDE from `http://www.vsdsp-forum.com/` and install it with command

wine vside_win32_v???.exe

where ??? is the version number of VSIDE. Remember use installation target c:\VSIDE\.

If everything went well, there should be VSIDE icon on your desktop. It is possible to start VSIDE from command line with

wine C:/VSIDE/VSIDE.EXE

If VSIDE was installed to default directory, Program Files (x86), spaces and parenthesis should be escaped. For example wine c:/Program\ Files\ \(x86\)/VSIDE/VSIDE.EXE which is rather horrible to write. Also some helper programs may have problems with special characters.

After installation, test toolchain and connections as instructed in section 4.2.

8.4 Manual Handling of the Flash Volume on Linux

Sometimes automation doesn't work or there are situations where better understanding is required about the system. This section tries to cover the details of mounting VS1005 Developer board external SPI flash.

When Developer board is booted with S1 pressed, dmesg shows information about newly added hardware. If dmesg shows nothing about VS1005 check PC - CN5 connection.

```
[4259172.620286] usb 1-3: new full-speed USB device number 7 using xhci_hcd
```

```
[4259172.749061] usb 1-3: New USB device found, idVendor=19fb, idProduct=ee02
[4259172.749063] usb 1-3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[4259172.749064] usb 1-3: Product: VS1005G1
[4259172.749065] usb 1-3: Manufacturer: Vlsi
[4259172.749066] usb 1-3: SerialNumber: 000000000001
[4259172.749463] usb-storage 1-3:1.0: USB Mass Storage device detected
[4259172.749520] scsi host8: usb-storage 1-3:1.0
[4259173.749339] scsi 8:0:0:0: Direct-Access     VLSI     VS1005-1         1.3  PQ: 0
[4259173.750102] sd 8:0:0:0: Attached scsi generic sg2 type 0
[4259173.750565] sd 8:0:0:0: [sdc] 480 4096-byte logical blocks: (1.92 MB/1.87 MiB)
[4259173.750737] sd 8:0:0:0: [sdc] Write Protect is off
[4259173.750746] sd 8:0:0:0: [sdc] Mode Sense: 00 00 00 00
[4259173.750906] sd 8:0:0:0: [sdc] Asking for cache data failed
[4259173.750916] sd 8:0:0:0: [sdc] Assuming drive cache: write through
[4259173.753225] sd 8:0:0:0: [sdc] 480 4096-byte logical blocks: (1.92 MB/1.87 MiB)
[4259173.761625]  sdc:
[4259173.762521] sd 8:0:0:0: [sdc] 480 4096-byte logical blocks: (1.92 MB/1.87 MiB)
[4259173.762845] sd 8:0:0:0: [sdc] Attached SCSI removable disk
```

Also lsusb shows the developer board

```
Bus 001 Device 007: ID 19fb:ee02
```

In the case that there is no filesystem, one must be created. No partitions are required.

```
sudo mkfs.vfat /dev/sdX -n vs1005_S
```

Change /dev/sdX to your device seen in dmesg. -n parameter is given to the command to name the volume as VS1005_S.

If the volume didn't automatically mounted, press S1 and reset the developer board.

Manually mounting is done with

```
sudo mount -t vfat /dev/sdX /some/target
```

and before resetting the board remember run

```
sudo umount /some/target
```

8.5 Features Which Don't Work on Linux

VSIDE on Wine hangs if a file which is part of the current solution is open in VSIDE and it is edited outside of VSIDE. So don't edit with your favorite editor and VSIDE same files.

9 Miscellaneous Tips

Some things are worth to remember and some things are not so obvious. This lists some things which is worth remembering and consider when developing VSOS application.

16 bits	The smallest size is 16 bits, sizeof(char) == sizeof(u_int16) == 1.
Stereo audio	Audio is always stereo. Always read, process and write both channels.
Spare the memory	Consider how much your functions reserve memory from the stack. Allocate dynamically and remember free if much memory is needed.
Sizes in printf	In printf there is no checks the size of parameter. Use %ld, %lu and %lx for 32-bit variables and %d, %u and %x for 16-bit.
Out of mem N	where N is number I-mem (0), X-mem (1), or Y-mem (2). Don't write beyond your arrays.

10 Latest Version Changes

Version 1.00, 2016-06-23

Initial version.

11 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