xCORE VocalFusion Control Users Guide

IN THIS DOCUMENT

- Introduction
- Prerequisities
- Controllable Entities
- Enabling Control in the Firmware
- Building the Command-line Utility
- Windows Driver Installation for USB control
- I^{2}C Control Setup on Raspberry Pi 3
- Using the command-line utility
- Making Changes Permanent
- Appendix

1 Introduction

This document describes how to use the provided command-line utilities to read and update the user configurable parameters on *xCORE VocalFusion* devices and evaluation kits.

The underlying mechanism for the control is provided by the *lib_device_control* library. This library implements host and device side APIs to provide acknowledged transport of control messages. A range of control transport mechanisms are available including USB, I^{2}C and xSCOPE. This allows changing of the *xCORE VocalFusion* parameters at runtime to meet specific application needs and available interfaces on the host.

For details on the architecture and implementation of parameter control within the firmware please refer to *VocalFusion Software Design Guide*[1]. The *XVF3000/3100 DSP Databrief*[2] details the DSP processing of the VocalFusion stack.

2 Prerequisities

- This document assumes familiarity with the XMOS xCORE architecture, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this guide are linked to in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary[3].

[1] http://www.xmos.com/published/vocalfusion-software-design-guide
[2] https://www.xmos.com/published/xvf3000/3100-dsp-databrief
[3] http://www.xmos.com/published/glossary

Publication Date: 2018/3/23 Document Number: XM012592A

3 Controllable Entities

The voice algorithm suite supports linear arrays to provide two stereo channels with up to 180° coverage and has been developed for far-field voice control in Stereo TV and/or video applications.

Each firmware variant exposes different runtime controllable parameters.

All *xCORE VocalFusion* firmware broadly divides its microphone processing into two logical blocks: Acoustic Echo Cancellation (AEC) and Beamforming & Post-processing (BAP). The *xCORE VocalFusion* device has two AEC blocks, one for each channel. Each block resides on a different tile and each has multiple control parameters associated with it. I/O control interface (such as I^2C) is connected to each logical block via xC interfaces, an array called `i_control` at time of writing. There are two in mono devices, AEC and BAP. There are three in stereo devices, AEC12, AEC34 and BAP.

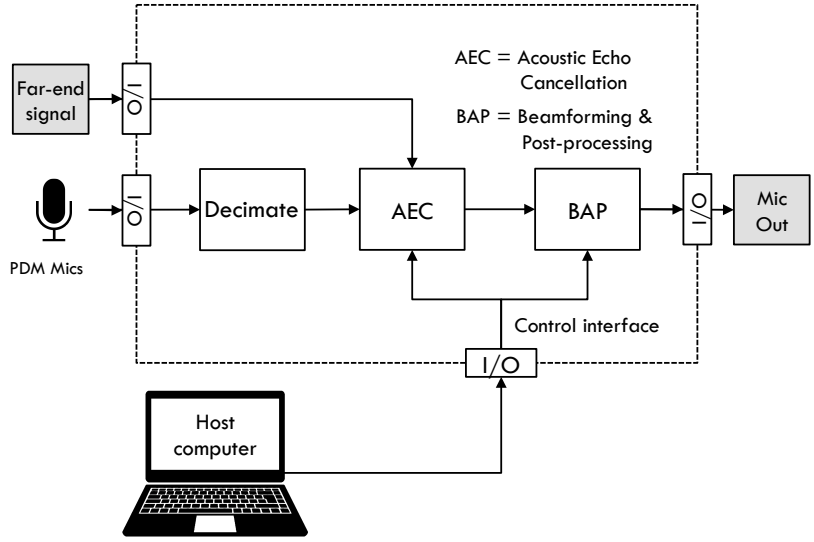

Figure 1: High level view of parameter control

A list of the available parameters that are runtime controllable can be found in the appendix in the following the tables:

- Commands for all firmware variants - params_both
- Additional commands for SmartTV only - params_smarttv

4 Enabling Control in the Firmware

If you are using a pre-compiled firmware binary, make sure it has control enabled. *xCORE VocalFusion* Speaker builds indicate `ctl` in the binary name, for example `1i2o2_cir43_usbctl` or `1i0o0_cir43_i2s_only_48khz_i2cctl`. *xCORE VocalFusion* Stereo builds have control enabled implicitly, for example `usb` or `i2s_master`.

The code for control is guarded by pre-processor macros. Figure 2 summarizes the macros required enable control over a particular transport. Note that it is typical to add these to the `Makefile` build configuration so that all source files within the project receive the define. The Makefile argument to do that is provided in the last column.

Figure 2: Enabling control in the firmware using preprocessor macros

Transport	Defines to set to 1	Makefile arguments
USB	BECLEAR_CONTROL_USB	`-DBECLEAR_CONTROL_USB=1`
I^2C	BECLEAR_CONTROL_I2C	`-DBECLEAR_CONTROL_I2C=1`
xSCOPE	BECLEAR_CONTROL_XSCOPE	`-fxscope -DBECLEAR_CONTROL_XSCOPE=1`

When enabling xSCOPE control, in addition to enabling the control code using `BECLEAR_CONTROL_XSCOPE`, the tools-provided xSCOPE library must be linked into the application using the `-fxscope` argument.

5 Building the Command-line Utility

XMOS provides example command-line utilities for controlling parameters inside the *xCORE VocalFusion* firmware.

The command-line utilities are supplied as source with Makefiles for building under multiple platforms. The utility may be built from the source using commonly available compilers. The following instructions provide step-by-step guides to building the binary/executable.

Open a xTIMEcomposer command-line window and navigate to the directory containing the command-line utility:

```
lib_xbeclear/host/control/
```

And then use the following to build:

Microsoft Visual Studio is required to compile the host control utility on Windows. It has been tested with Microsoft Visual Studio Community 2015 (Version 14.0.23107.0 D14REL).

▶ `nmake /f Makefile.Win32`

The above command will build two binaries `vfctrl_xscope` and `vfctrl_usb` which support xSCOPE and USB control transports.

Xcode is required to compile the host utility on macOS.

▶ `make -f Makefile.OSX`

The above command will build two binaries `vfctrl_xscope` and `vfctrl_usb` which support xSCOPE and USB control transports.

g++ is required to compile the host utility on Linux.

For linux x86 hosts:

▶ `make -f Makefile.Linux64`

The above command will build two binaries `vfctrl_xscope` and `vfctrl_usb` which support xSCOPE and USB control transports.

For linux ARM running on a Raspberry Pi 3:

▶ `make -f Makefile.Pi`

The above command will build two binaries `vfctrl_usb` and `vfctrl_i2c` which support USB and I^2C control transports.

Linux and Pi builds list dependencies in their respective makefiles. These can be installed with by doing *apt-get install* on Debian derived systems. Typically, these are libusb, libreadline and libncurses5 with headers. Please refer to the makefile itself for details.

6 Windows Driver Installation for USB control

The *xCORE VocalFusion* device with USB host connection is a composite USB device which requires a driver to be installed for the control interface when used with Windows. The driver is installed by locating the device in Device manager. Select **Control Panel ▶ Device Manager**. For other OS's or embedded hosts this step is not required.

When you plug in the board on a fresh system, two unknown devices will appear, *XMOS Control* and *XMOS DFU*. Right click on *XMOS Control* and select **Update Driver Software...**.

Select **Browse for driver software on your computer**, or equivalent, and select the `lib_xbeclear/host/control/libusb/Win32/driver` folder.

The driver should succesfully install and create a device called *XMOS Microphone Array Control*.

Note that the *XMOS DFU* device remains unrecognized at this point. DFU host utilities are available for Raspberry Pi, Linux and macOS, please see *xCORE VocalFusion software design guide*.

The Windows host is now ready to send control requests to the device.

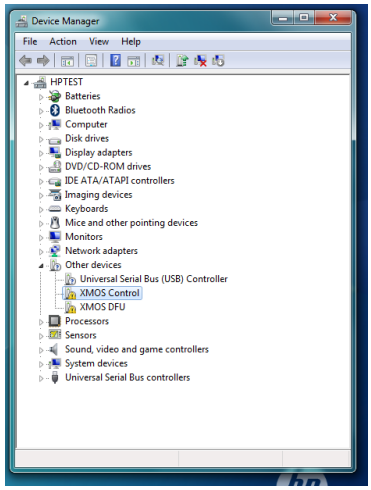

Figure 3: Device before driver install

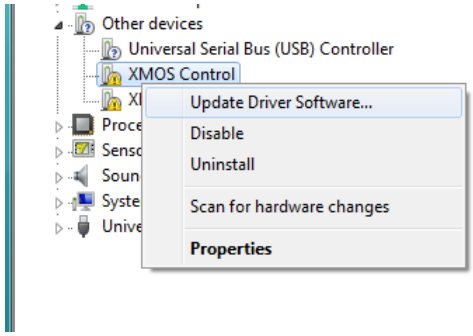

Figure 4: Update the driver

6.1 Windows Driver Issues

1. If the driver installation hangs (seen on Windows 7):

   Disable checking the driver store. **Control Panel ▶ Change device installation settings ▶ No let me choose what to do ▶ Never install driver software from Windows Update**

2. If Windows Security says it can't verify the publisher of the driver:

   Select **Install this driver software anyway**. This often results in a working driver on Windows 7.

3. There is a known issue with the Windows device driver where, on systems running Windows 10 Anniversary Update and newer, the driver will not install correctly. This is because the driver signing method used is not the new attestation signing. The issue is being reviewed. In the mean-time, disabling driver signing checking allows the driver to install.

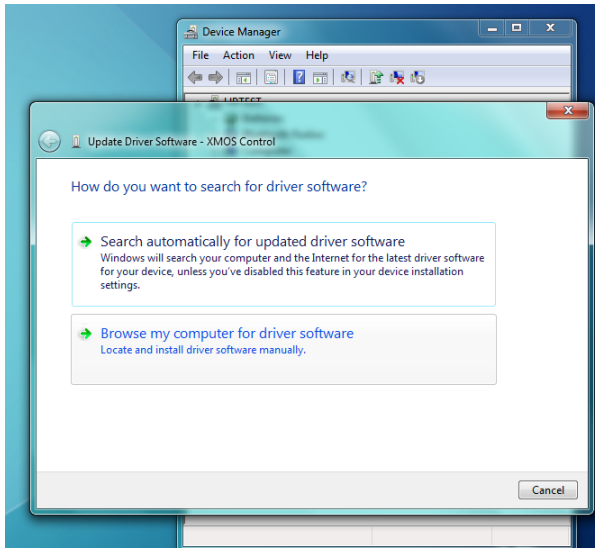

Figure 5: Locate the driver

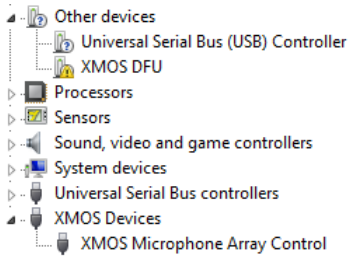

Figure 6: Once the driver has been installed

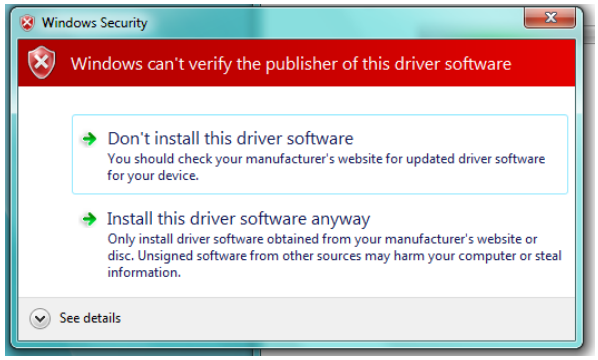

Figure 7: Windows Security

7 I²C Control Setup on Raspberry Pi 3

It is possible to control device parameters via I²C, the device being an I²C slave at bus address 0x2c. It is recommended this is evaluated using a Raspberry Pi 3.

The I²C control implementation requires support for clock stretching. This is because *xCORE VocalFusion* will service control requests in between sample blocks (typically 16ms long, 256 samples at 16kHz). It also requires support for repeated starts where two back-to-back transactions are concatenated with only a single stop bit at the end of the last transfer. Repeated starts are used to cluster two parts of a read request together in the provided host control utility.

Raspberry Pi 3 has a built in I²C peripheral inside the BCM2837 and can be made to support repeated starts. There is however a chip bug[4] in the clock stretching implementation. It means that it may miss clock line transitions and cause some transactions to fail.

In order to workaround the Raspberry Pi 3 I²C bug it is recommended to use the i2c-gpio module built into recent versions of Raspbian, from around June 2016 Jessie. This driver, which uses bit banging to emulate an I²C host, fully supports clock stretching and repeated starts by default. The below instructions describe how to replace the hardware I²C driver with the bit-banged I²C driver and conveniently instantiate the new I²C driver on the same bus ID using the same physical GPIO pins as the hardware driver. This means that its use over hardware I²C is transparent.

To avoid recompilation of the Linux kernel a loadable kernel module is used to insert the new driver into the OS. A full description of how to do this can be found in the link below[5], however a brief summary of the required steps, found to work on a clean copy of Debian Jessie shipped with `Noobs` is provided in the instructions below:

7.1 Update Raspbian

First, update the Raspbian Linux distribution to the latest version (version 4.4.26-v7+ tested). This may take a few hours:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

Before proceeding with the next steps, ensure you have re-booted your Raspberry Pi 3 so that it is running the latest version of Raspbian that you have just downloaded. This can be done using the following command:

```
sudo reboot
```

[4] https://www.advamation.com/knowhow/raspberrypi/rpi-i2c-bug.html
[5] http://www.thegeekstuff.com/2012/04/linux-lkm-basics/

7.2 Disable Native I²C

Now you have an updated OS, you should disable the hardware I²C on your Raspberry Pi 3, which is a straightforward operation using device tree. To do this, edit the file /boot/config.txt to remove the comment around line 46 and set it as follows:

```
dtparam=i2c_arm=off
```

You will need to reboot for this change to take effect.

7.3 Build Kernel Module for Bitbanged I²C

Please ensure about a gigabyte of available SD card space for the below steps.

Next, download the kernel source (this may also take a long time to download and extract). The source download script requires the installation of two helper applications that are not installed by default. The installation of these is included below:

```
sudo apt-get install bc
sudo apt-get install libusb-1.0-0-dev libreadline-dev libncurses5-dev
git clone http://github.com/notro/rpi-source
cd rpi-source
python rpi-source
cd ..
```

For further information on building and loading modules under Linux the link below provides a useful reference[6].

If you need to rerun the above script, the original directory as well as `linux` symbolic link need to be deleted.

Next you need to download the kernel module loader source which presents the driver as a loadable module to the OS:

```
git clone https://github.com/kadamski/i2c-gpio-param.git
cd i2c-gpio-param
make
```

If the *git clone* command doesn't work, clone the module directly from the URL above as a zip file, extract it and rename the folder as i2c-gpio-param.

On running `make` you should see a console output similar to this:

[6] http://www.cyberciti.biz/tips/build-linux-kernel-module-against-installed-kernel-source-tree.html

```
pi@raspberrypi:~/i2c-gpio-param $ make
make -C /lib/modules/`uname -r`/build M=/home/pi/i2c-gpio-param modules
make[1]: Entering directory '/home/pi/linux-1315
  ↪ ab6319b02a436a6c1a2b38608e52b94d22fb'
CC [M]  /home/pi/i2c-gpio-param/i2c-gpio-param.o
Building modules, stage 2.
MODPOST 1 modules
CC      /home/pi/i2c-gpio-param/i2c-gpio-param.mod.o
LD [M]  /home/pi/i2c-gpio-param/i2c-gpio-param.ko
make[1]: Leaving directory '/home/pi/linux-1315
  ↪ ab6319b02a436a6c1a2b38608e52b94d22fb'
```

7.4 Load Kernel Module

And finally the module needs inserting into the kernel and the driver needs instantiating with the correct parameters. For help regarding use of the bit-banged I^2C module please checkout the documentation at the i2c-gpio-param repo[7].

Save the following text in the script file `load_i2c-gpio_driver.sh` in your home directory:

```
# start i2c-gpio directory
cd /home/pi/i2c-gpio-param
# load the i2c bit banged driver
sudo insmod i2c-gpio-param.ko
# instantiate a driver at bus id=1 on same pins as hw i2c with 60s timeout
sudo bash -c 'echo "1 2 3 5 6000 0 0 0" > /sys/class/i2c-gpio/add_bus'
# remove the default i2c-gpio instance
sudo bash -c 'echo "7" > /sys/class/i2c-gpio/remove_bus'
# return to previous dir
cd /home/pi/
```

Now you can simply run the newly created shell script to enable the bit-banged I^2C driver:

```
sh load_i2c-gpio_driver.sh
```

7.5 Set Up the Hardware

You should then attach the pins in the following manner (see also Figure 8). The full pinout of the Raspberry Pi 3 header can be viewed from the link below[8].

- Attach the Raspberry Pi SDA pin 3 (BCM2) to pin 9 (X0D24) of Expansion Header J5.
- Attach the Raspberry Pi SCL pin 5 (BCM3) to pin 12 (X0D25) to Expansion Header J5.

[7] https://github.com/kadamski/i2c-gpio-param
[8] https://pinout.xyz/pinout/i2c

- Ground should also be shared - available on Raspberry Pi pin 6 to one of pins 2, 6, 8, 14, 15, 16 of Expansion Header J5.

Now run the command:

```
i2cdetect -y 1
```

to check that the Raspberry Pi host can see the device before running the host application. It should show a device at address 0x2c. If it does not check your wiring between the Raspberry Pi and the *xCORE VocalFusion* board and ensure you are running a version of firmware with I^{2}C control enabled (i2cctl is contained in the binary name).

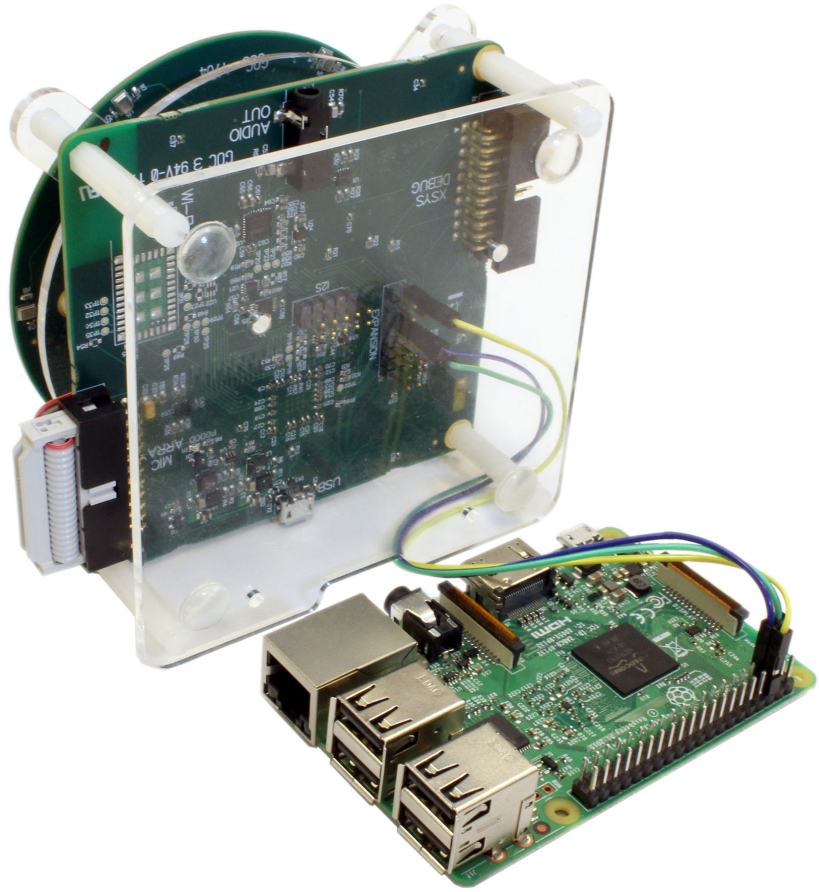

Figure 8: Raspberry Pi 3 connection for I^{2}C control

Following the instructions given so far will enable the bit-banged I^{2}C driver in the current session. To make it available automatically after each reboot, it is necessary

to load the module at boot time. You can do this by adding a boot-time cron job to run the above shell script. The script can auto-run at boot time by adding the following line in the editor window that appears after typing `sudo crontab -e`:

```
@reboot sh /home/pi/load_i2c-gpio_driver.sh
```

8 Using the command-line utility

Depending on the transport used for control the binaries have either the suffix _usbctl, _xscopectl or _i2cctl for USB, xSCOPE and I^2C control respectively. It should be noted that the xSCOPE binary must be run on the device in the following manner:

```
xrun --xscope-port localhost:10101 <file_name>.xe
```

Use `xrun` alone with other control transports, or use xflash.

Once this program is running, you can then start the host application.

in Windows:

- `bin/vfctrl_usb.exe`

in macOS:

- `./bin/vfctrl_usb`

in Linux:

- `sudo ./bin/vfctrl_usb`

The following udev rule (belongs in `/etc/udev/rules.d`) should allow running the USB utility as a regular user. See customdefines.h to confirm your product ID. Most configurations are USB audio class 1.0 and have PID 0x11. Class 2.0 configurations should have PID 0x10.

```
ATTR{idVendor}=="20B1", ATTR{idProduct}=="0011"
GROUP="plugdev"
```

The xSCOPE command-line utility must be run with the IP address and port as second and third command-line parameter, respectively. For example in a Windows machine using a local port 10101:

```
./bin/vfctrl_xscope.exe localhost 10101
```

8.1 Read a control parameter

If no further arguments are added then the control utility will read the specified parameter and print the current value. For example:

```
$ ./bin/vfctrl_usb RT60
RT60:0.9
```

In case of Stereo SmartTV devices, AEC-related parameters will report two parameters, one for each AEC block, for example:

```
$ ./bin/vfctrl_usb RT60
RT60:0.9 0.9
```

8.2 Write a control parameter

If you wish to write to a parameter then add the value to be written as the last argument to the command line:

```
$ ./bin/vfctrl_usb AGCONOFF 0
AGCONOFF:0
```

Differently from the reading operations, the AEC commands for Stereo SmartTV devices accept only one value to be written, so that the same value is applied to both of the AEC blocks.

8.3 List control parameters

If the parameter can not be found or the value requested to be written is outside of bounds then an error message will be shown. To obtain a list of controllable parameters type:

```
$ ./bin/vfctrl_usb parameters
```

This will display available parameters and related details, for example:

```
paramater              type    max          min          r/w          info
---------              ----    ---          ---          ---          ----
AECFREEZEONOFF         int     1            0            read-write   ...
AECNORM                float   16           0.25         read-write   ...
AECPATHCHANGE          int     1            0            read-only    ...
RT60                   float   0.9          0.25         read-only    ...
...
```

Further information can be found by typing the `--help` command:

```
$ ./bin/vfctrl_usb --help
Connected to a Linear Stereo device.
Usage:

 <parameter>                Return the current value of <parameter>
 <parameter> <value>        Assign <value> to <parameter>
 --help (-h)                Display this information
 --help (-h) <parameter>    Display information for <parameter>
 parameters (-p)            Display the list of parameters
 tuning_cmds (-tc)          Display the list of tuning commands
```

8.4 Troubleshooting

- The commands `AGCTIME` and `AGCGAIN` are `read-write`, but the values read back are normally different from the ones written by the user. The value of the `AGCTIME` is used to calculate an internal AGC coefficient, so reading the parameter back returns this internal AGC coefficient. There is currently no defined meaning to the internal `AGCTIME` coefficient value, so it can be ignored. The value of the `AGCGAIN` changes and essentially represents the current state of AGC. By definition it will most of the time read back a different value than the write value.
- *xCORE VocalFusion* board in I^2S configuration requires running I^2S clocks for I^2C control to work. Configurations that are I^2S master will always have running clocks. Configurations that are I^2S slave need to have a master present supplying the clocks.
- If `i2cdetect -y 1` shows a device present but `vfctrl_i2c` returns the error *rdwr ioctl error -1: No such device or address*, it is a good idea to check that an I^2C based *xCORE VocalFusion* device is supplied with valid clocks. Specifically a valid MCLK, BCLK and LRCLK signal from the I^2S master. Control requests can only be serviced when the device is properly clocked.

9 Making Changes Permanent

Once the control mechanism has been used to evaluate the various settings you may want to make “permanent” changes to the default parameter settings.

This should be done in the *xCORE VocalFusion* source code, typically in the `beclear_conf.h` header filer. Further details can be found in the Software Design Guide.

10 Appendix

10.1 Resource utilization

Enabling control within the firmware will utilize additional chip resources. The amount of extra resources utilized depends on the transport chosen and the existing *xCORE VocalFusion* functionality. For example, adding control over USB to a USB Audio connected configuration may consume an additional 5KB of memory, 3 channel ends and no extra processing resource. In contrast, adding control over

I^2C to an I^2S connected configuration may consume an additional 9KB of memory, 5 channels ends and one extra logical core needed to implement the I^2C slave peripheral and control decoding.

10.2 Controllable parameters

The tables below detail the control parameters supported in *xCORE VocalFusion*. There are two functional groups: AEC (Adaptive Echo Cancellation) and BF-BP/BAP (BeamForming and Post processing).

You may also view this list by running the command utility with the `parameters` argument.

The user configurable pre-processing parameters are listed in Table 1, the configurable AEC parameters are listed in Table 2 and the configurable Beamformer and Post Processor parameters are listed in Tables 3 and 4.

Table 1: Parameter descriptions for the pre-processing (linear and circular arrays)

parameter	type	range	description
MIC_ATTEN	APES_INT	[-100,0]	MIC input signal attenuator in decibels (read-write) (Default 0dB).
AEC_REF_ATTEN	APES_INT	[-100,0]	AEC reference signal attenuator in decibels (read-write) (Default: 0dB).

Table 2: Parameter descriptions for the AEC module (linear and circular arrays)

parameter	type	range	description
AECFREEZEONOFF	APES_INT	[0,1]	Adaptive Echo Canceler updates inhibit (read-write). 0 = Adaptation enabled (default) 1 = Freeze adaptation, filter only
AECNORM	APES_FLOAT1	$[0.25 .. 16.0]$	Limit on norm of AEC filter coefficients (read-write). (default: 2.0)
AECSILENCELEVEL	APES_FLOAT1	$[0.0 .. 1.0]$	Threshold for signal detection in AEC (read-write). $[-\infty .. 0]$ dBov (default: -80dBov $\approx 10\log_{10}(1 \cdot 10^{-8})$)
AECSILENCEMODE	APES_INT	[0,1]	AEC far-end silence detection status (read-only). 0 = false (signal detected) 1 = true (silence detected)
HPFONOFF	APES_INT	[0..3]	High-pass Filter on microphone signals (read-write). 0 = OFF 1 = ON - 70 Hz cut-off (default) 2 = ON - 125 Hz cut-off 3 = ON - 180 Hz cut-off
RT60	APES_FLOAT1	$[0.250 .. 0.900]$	Current RT60 estimate in seconds (read-only). (default: 0.500)
RT60ONOFF	APES_INT	[0,1]	RT60 Estimation for AES (read-write). 0 = OFF 1 = ON (default)

parameter	type	range	description
AGCDESIREDLEVEL	APES_FLOAT1	$[0.0 .. 1.0]$	Target power level of the output signal (read-write). $[-\infty .. 0]$ dBov (default: -23 dBov $\approx 10\log_{10}(0.005)$)
AGCGAIN	APES_FLOAT1	$[1.0 .. 1000.0]$	Current AGC gain factor (read-write). $[0 .. 60]$ dB (default: 0.0 dB $= 20\log_{10}(1.0)$)
AGCONOFF	APES_INT	[0,1]	Automatic Gain Control (read-write). 0 = OFF 1 = ON (default)
AGCMAXGAIN	APES_FLOAT1	$[1.0 .. 1000.0]$	Maximum AGC gain factor (read-write). $[0 .. 60]$ dB (default 30 dB $\approx 20\log_{10}(31.6)$)
AGCTIME	APES_FLOAT1	$[0.1 .. 1.0]$	Ramp-up/down time-constant in seconds(read-write). $[0.1 .. 1.0]$ (default: 0.9 s)
CNIONOFF	APES_INT	[0,1]	Comfort Noise Insertion (read-write). 0 = OFF 1 = ON (default)
ECHOONOFF	APES_INT	[0,1]	Echo suppression (read-write). 0 = OFF 1 = ON (default)
FREEZEONOFF	APES_INT	[0,1]	Adaptive beamformer and postprocessor updates (read-write). 0 = Adaptation enabled (default) 1 = Freeze adaptation, filter only
GAMMA_NN	APES_FLOAT1	$[0.0 .. 3.0]$	Over-subtraction factor of non-stationary noise (read-write). min .. max attenuation (default: 1.1)
GAMMA_NS	APES_FLOAT1	$[0.0 .. 3.0]$	Over-subtraction factor of stationary noise (read-write). min .. max attenuation (default: 1.0)
MIN_NN	APES_FLOAT1	$[0.0 .. 1.0]$	Gain-floor for non-stationary noise suppression (read-write). $[-\infty .. 0]$ dB (default: -10 dB $\approx 20\log_{10}(0.3)$)
MIN_NS	APES_FLOAT1	$[0.0 .. 1.0]$	Gain-floor for stationary noise suppression (read-write). $[-\infty .. 0]$ dB (default: -16 dB $\approx 20\log_{10}(0.15)$)
NONSTATNOISEONOFF	APES_INT	[0,1]	Non-stationary noise suppression (read-write). 0 = OFF 1 = ON (default)
STATNOISEONOFF	APES_INT	[0,1]	Stationary noise suppression (read-write). 0 = OFF 1 = ON (default)
TRANSIENTONOFF	APES_INT	[0,1]	Transient echo suppression (read-write). 0 = OFF 1 = ON (default)

Table 3: Parameter description for the BF and PP (linear and circular arrays)

parameter	type	range	description
FSBPATHCHANGE	APES_INT	[0,1]	FSB Path Change Detection (read-only). 0 = false (no path change detected) 1 = true (path change detected)
FSBUPDATED	APES_INT	[0,1]	FSB Update Decision (read-only). 0 = false (FSB was not updated) 1 = true (FSB was updated)
GAMMA_E	APES_FLOAT1	[0.0 .. 3.0]	Over-subtraction factor of echo (direct and early components) (read-write). min .. max attenuation (default: 1.0)
GAMMA_ENL	APES_FLOAT1	[0.0 .. 5.0]	Over-subtraction factor of non-linear echo (read-write). min .. max attenuation (default: 1.0)
GAMMA_ETAIL	APES_FLOAT1	[0.0 .. 3.0]	Over-subtraction factor of echo (tail components) (read-write). min .. max attenuation (default: 1.0)
NLAEC_MODE	APES_INT	[0..2]	Non-Linear AEC training mode (read-write). 0 = OFF (default) 1 = ON - phase 1 2 = ON - phase 2
NLATTENONOFF	APES_INT	[0,1]	Non-Linear echo attenuation (read-write). 0 = OFF (default) 1 = ON
VOICEACTIVITY	APES_INT	[0,1]	Signal energy exceeded a threshold (read-only). 0 = false (no voice activity) 1 = true (voice activity)
DOAANGLE	APES_INT	[0 .. 359]	DOA angle; current value, orientation depends on build configuration (read-only).
KEYWORDDETECT	APES_INT	[0,1]	Keyword detected; current value, not sticky (read-only). 0 = not detected 1 = detected
DOAANGLE	APES_INT	[0 .. 359]	DOA angle; current value, orientation depends on build configuration (read-only).

Table 4: Parameter description for the BF and PP, linear and circular arrays (continued)

When using **Circular Arrays** a number of additional user configurable parameters are available. Tables 5 details the additional AEC parameters and Table 6 details the additional Beamformer and Post Processor parameters.

Table 5: Additional AEC parameters for circular arrays

parameter	type	range	description
AECPATHCHANGE	APES_INT	[0,1]	AEC Path Change Detection (read-only). 0 = false (no path change detected) 1 = true (path change detected)

parameter	type	range	description
GAMMA_NN_SR	APES_FLOAT1	[0.0 .. 3.0]	Over-subtraction factor of non-stationary noise for ASR (read-write). min .. max attenuation (default: 1.1)
GAMMA_NS_SR	APES_FLOAT1	[0.0 .. 3.0]	Over-subtraction factor of stationary noise for ASR (read-write). min .. max attenuation (default: 1.0)
GAMMAVAD_SR	APES_FLOAT1	[0.0 .. 1000.0]	Set the threshold for voice activity detection (read-write). $[-\infty$ .. 60] dB (default: 3.5 dB $\approx 20\log_{10}(1.5)$)
MIN_NN_SR	APES_FLOAT1	[0.0 .. 1.0]	Gain-floor for non-stationary noise suppression for ASR (read-write). $[-\infty$ .. 0] dB (default: -10 dB $\approx 20\log_{10}(0.3)$)
MIN_NS_SR	APES_FLOAT1	[0.0 .. 1.0]	Gain-floor for stationary noise suppression for ASR (read-write). $[-\infty$ .. 0] dB (default: -10 dB $\approx 20\log_{10}(0.3)$)
NONSTATNOISEONOFF_SR	APES_INT	[0,1]	Non-stationary noise suppression for ASR (read-write). 0 = OFF 1 = ON (default)
SPEECHDETECTED	APES_INT	[0,1]	Speech-like signal detected (internal, read-only). 0 = false (no speech detected) 1 = true (speech detected)

Table 6: Additional BF and PP parameters for circular arrays

When using **Linear Arrays** a number of additional user configurable parameters are available. Tables 7 details the additional AEC parameters and Table 8 details the additional Beamformer and Post Processor parameters.

parameter	type	range	description
AECERLMAX	APES_FLOAT1	[1.0 .. 99856.0]	maximum erl estimate (write-only). (default: 99000.0)
MAX_RT60	APES_FLOAT1	[0.0 .. 0.9]	Set the upper limit for the revest T60 estimator in seconds (write-only). (default: 0.9)
AEC_REF_DELAY	TYPE_INT	[0 .. 2400]	"Parametric delay for the AEC reference samples. Value is given in samples at 16kHz. (default: 0)

Table 7: Additional AEC parameters for linear arrays

parameter	type	range	description
BEAMANGLE	APES_FLOAT1	$[-1.0 .. 1.0]$	Center of the beam for desired speech sources (read-write). $[-90° .. 90°]$ (default: $0° = \sin^{-1}(0.0)\frac{360}{2\pi}$)
BEAMWIDTH	APES_FLOAT1	$[0.2 .. 1.0]$	Width of the beam for desired speech sources (read-write). $[23° .. 180°]$ (default: $60° \approx \sin^{-1}(0.5)\frac{360}{\pi}$)
FSBFREEZEONOFF	APES_INT	[0,1]	Adaptive beamformer updates (read-write). 0 = Adaptation enabled (default) 1 = Freeze adaptation, filter only
SPTHRESH	APES_FLOAT1	$[0.0 .. 1.0]$	Set parameter value for DNNS (read-write). (default: 0.0065)
SR_ABSQFLOOR	APES_FLOAT	$[0.0 .. 1000.0]$	Absolute noise floor for voice activity detection (read-write). $[-\infty .. 60]$ dB (default: $-\infty$ dB $= 20\log_{10}(0.0)$)
SR_GAMMA_NN	APES_FLOAT1	$[0.0 .. 3.0]$	Gain-floor for non-stationary noise suppression (read-write). min .. max attenuation (default: 1.1)
SR_GAMMA_NS	APES_FLOAT1	$[0.0 .. 3.0]$	Over-subtraction factor of stationary noise (read-write). min .. max attenuation (default: 1.0)
SR_GAMMA_VAD	APES_FLOAT1	$[0.0 .. 1000.0]$	Threshold for voice activity detection (read-write). $[-\infty .. 60]$ dB (default: 23.5 dB $\approx 20\log_{10}(15)$)
SR_MIN_NN	APES_FLOAT1	$[0.0 .. 1.0]$	Gain-floor for non-stationary noise suppression (read-write). $[-\infty .. 0]$ dB (default: -10 dB $\approx 20\log_{10}(0.3)$)
SR_MIN_NS	APES_FLOAT1	$[0.0 .. 1.0]$	Gain-floor for stationary noise suppression (read-write). $[-\infty .. 0]$ dB (default: -16 dB $\approx 20\log_{10}(0.15)$)
SR_NONSTATNOISEONOFF	APES_INT	[0,1]	Non-stationary noise suppression for ASR (read-write). 0 = OFF 1 = ON (default)
SR_STATNOISEONOFF	APES_INT	[0,1]	Stationary noise suppression for ASR (read-write). 0 = OFF 1 = ON (default)
XNLTRAINONOFF	APES_INT	[0,1]	Non-linear matrix training (read-write). 0 = OFF(default) 1 = ON

Table 8: Additional BF and PP parameters for linear arrays

10.2.1 References

xCORE-200: The XMOS XS2 Architecture

https://www.xmos.com/published/xs2-isa-specification

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide