XMOS XVF3800 - User Guide

Release: 2.0.0
Publication Date: 2023/07/03

XMOS

Table of Contents

1 Overview

The XMOS VocalFusion ® XVF3800 is a high-performance voice processor that uses microphone array processing and a sophisticated audio processing pipeline to capture clear, high-quality speech from anywhere in a room. The XVF3800 uses the XMOS xcore.ai processor and supports a range of integrated and accessory voice communication applications.

This document discusses:

- Setting up the hardware,
- Using the Host application,
- Tuning the XVF3800 firmware,
- Building and deploying an XVF3800 executable image,
- Some acoustic design guidelines, and
- Tools for device firmware upgrade (DFU).

It also includes a list of control commands in an appendix.

2 Setting Up the Hardware

2.1 Introduction

This section explains the process of setting up and configuring the XVF3800 firmware on an XK-VOICE-SQ66 development kit.

Version v2.0.0 of the XVF3800 firmware supports audio I/O via I^{2}S or USB UAC2. The instructions below describe how to set up an evaluation system in either configuration.

Note: The XK-VOICE-SQ66 development kit is intended for use as an evaluation board for the XVF3800 and after power on or reset the device will stop processing audio after 8 hours of continuous use. The board must be restarted after 8 hours to resume operation. Licensed production XVF3800 devices do not have this restriction.

2.2 Hardware Setup - USB (UA) Configuration)

The `UA` configuration is the simplest configuration as the XK-VOICE-SQ66 development kit can be connected directly to a computer using a USB port.

2.2.1 UA - Required Components

- An XK-VOICE-SQ66 development kit board.
- An XMOS XTAG4 with associated ribbon cables (Provided in the XK-VOICE-SQ66 development kit package).
- A setup or development machine (Windows, Mac OS or Linux are supported). This machine must have at least two USB connections.
- A micro USB cable to connect the setup machine to the XTAG4 and a second micro USB cable to connect the XK-VOICE-SQ66 development kit to the development machine.

2.2.2 Setting up the Evaluation System in UA Configuration

The following steps are required to set up the XVF3800 evaluation hardware:

1. On the development computer, install the latest available *XTC Tools*, available from https://www.xmos.ai/software-tools/. Installation instructions for the supported platforms are available at https://www.xmos.ai/view/Tools-15-Documentation.

2. Connect the USB port on the XK-VOICE-SQ66 development kit to the development computer using a USB cable. (This provides power to the device.)

3. Connect an XTAG-4 debug adapter to the setup computer via USB, and connect it to the XK-VOICE-SQ66 development kit using the supplied ribbon cable. The cable should be plugged into XSYS2 connector on the XK-VOICE-SQ66 development kit. Fig. 2.1 below shows the UA configuration of the XK-VOICE-SQ66 development kit
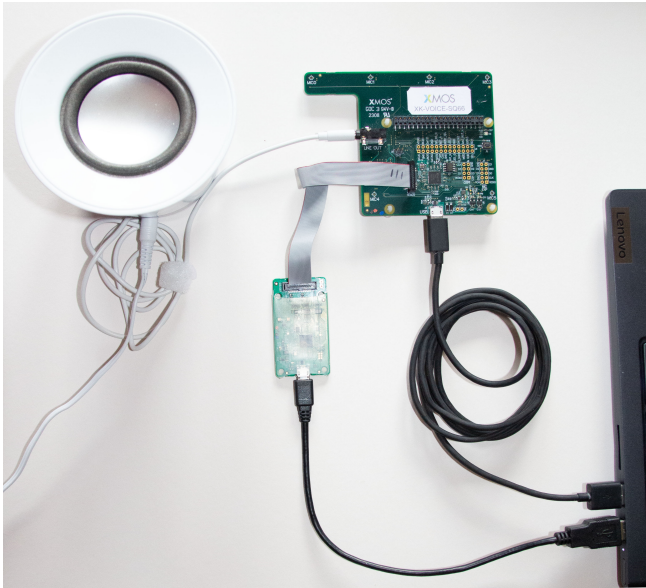
Fig. 2.1: XK-VOICE-SQ66 development kit in UA configuration

4. Open an XTC tools terminal window on the development computer. Verify that the XTAG4 has been correctly connected by running the following command in that window

```
xflash -l
```

The output from this should be of the following form:

```
Available XMOS Devices
----------------------

  ID    Name            Adapter ID  Devices
  --    ----            ----------  -------
  0     XMOS XTAG-4     ABCDE123    P[0]
```

Note: If the XTAG-4 is not properly connected to the development machine, then `xflash` will report `No Available Devices Found`. If the XK-VOICE-SQ66 development kit is not properly connected to the XTAG-4, then the Devices column will read `None`. For further guidance on the use of the *XTC tools*, see the https://www.xmos.ai/view/Tools-15-Documentation.

2.2.3 Installing the UA firmware

The XVF3800 firmware release package provided contains several precompiled binaries.

The provided binary images have names in the format:

```
application_xvf3800_ua[configuration-options].xe
```

where the [configuration-options] is constructed as detailed in Table 2.2 below.

Table 2.1: Build configuration settings

Option	Values	Description
I2S Sample rate	-lr16 / -lr48	to choose between a 16 kHz or 48 kHz I^{2}S LR clock (and therefore reference input and processed output audio sample rate). This setting is only used in INT configurations
USB sample rate	-io16 / -io48	to choose between a 16 kHz or 48 kHz rate for the USB rate. Input and output rates must be the same. This setting is only used in UA configurations
Microphone topology	-sqr / -lin	to choose between a "squarecular" or linear microphone array
Control protocol	-i2c / -spi	to choose between I^{2}C and SPI control modes. This setting is only used in INT configurations, the UA configurations always use USB control mode
Audio MCLK	-extmclk	to choose between external or internal MCLK. This setting is only used in INT configurations

Select the required binary firmware image from the release package, and transfer it to the XK-VOICE-SQ66 development kit using the `xflash` tool.

```
xflash application_xvf3800_ua[...].xe
```

The XTAG4 adapter can be disconnected from the XK-VOICE-SQ66 development kit after successfully installing the firmware image if desired.

After completing the initial setup, follow the instructions in the section *Setting up the Audio* below.

2.3 Hardware Setup - I2S (INTDEV) Configuration

The following instructions explain how to set up a Raspberry Pi computer with the XK-VOICE-SQ66 development kit in order to evaluate the operation of the XVF3800 using an I^{2}S audio interface in the `intdev` configuration.

2.3.1 INTDEV Required Components

- An XK-VOICE-SQ66 development kit board.

- An XMOS XTAG4 with associated ribbon cables (Provided in the XK-VOICE-SQ66 development kit package).

- A setup or development machine (Windows, Mac OS or Linux are supported). This must support USB connections and have the ability to write onto SD memory cards.

- A USB cable to connect the setup machine to the XTAG4.

- A Raspberry Pi microcomputer; either a Raspberry Pi 3 Model B or 4 Model B will work for this evaluation. See https://www.raspberrypi.com/ for more information.

- A region-appropriate USB power supply is also required. The XK-VOICE-SQ66 development kit obtains its power supply from the Raspberry Pi.

- An SD memory card - minimum 8GB size

- A stacking header block, such as pictured in Fig. 2.2 to mount the XK-VOICE-SQ66 development kit board onto the Raspberry Pi using the standard 40 pin GPIO header. A suitable part is Toby REF-182668-01 .
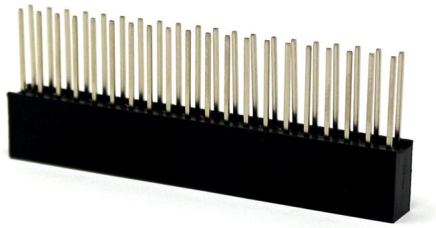
Fig. 2.2: Raspberry Pi HAT Connector - 10 mm Extended Tail Socket

- The XVF3800 evaluation firmware binary release package (available from https://xmos.ai or via an XMOS representative).

2.3.2 Setting up the INTDEV configuration

The following steps are required to set up the XVF3800 evaluation hardware:

1. On the setup machine, install the latest available *XTC Tools*, available from https://www.xmos.ai/software-tools/. Installation instructions for the supported platforms are available at https://www.xmos.ai/view/Tools-15-Documentation.

2. On the setup machine, install and run the Raspberry Pi Imager, available from https://www.raspberrypi.com/software

3. Using the Raspberry Pi imager, copy the latest available 32 bit version of the Raspberry Pi OS onto an SD card.

4. Insert the programmed SD card into the Raspberry Pi. Attach any required peripherals (keyboard, mouse, monitor), connect the power supply and follow the setup prompts, but skip the option to update the operating system software.

5. Once the Pi has booted, open a terminal window and download the `vocalfusion-rpi-setup` utility from https://github.com/xmos/vocalfusion-rpi-setup which should be saved in a convenient directory on the Raspberry Pi. The following commands will setup the Raspberry Pi for use with the XK-VOICE-SQ66 development kit:

```
git clone https://github.com/xmos/vocalfusion-rpi-setup
cd vocalfusion-rpi-setup
./setup.sh xvf3800-intdev
```

To enable a remote GUI access on the Raspberry Pi, the VNC service should be enabled at this point with the following command.

```
sudo raspi-config
```

Select `3 Interfaces Options` and enable VNC and SSH (if not already set using the imager) on the next screen.

6. Shutdown the Pi, detach the power and mount the XK-VOICE-SQ66 development kit board onto the Raspberry Pi header. After mounting the board, reattach power and verify the Raspberry Pi restarts.

   The evaluation hardware is now ready to use, and should resemble Fig. 2.3.
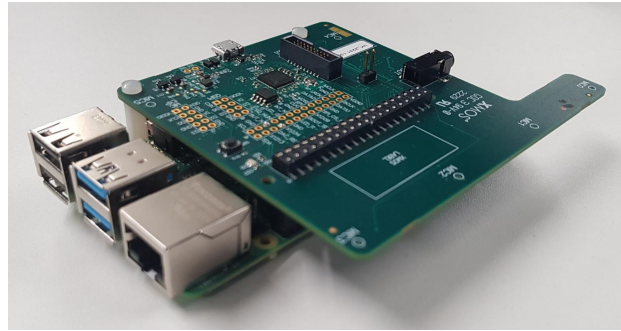
Fig. 2.3: XVF3800 evaluation kit

2.3.3 Installing the INTDEV Firmware

The XVF3800 firmware release package provided contains several precompiled binaries.

The provided binary images have names in the format:

```
application_xvf3800_[device-configurations][configuration-options].xe
```

where the [device-configurations] can be *intdev* or *ua* where the [configuration-options] is constructed as detailed in Table 2.2 below.

Table 2.2: Build configuration settings

Option	Values	Description
I2S Sample rate	-lr16 / -lr48	to choose between a 16 kHz or 48 kHz I^{2}S LR clock (and therefore reference input and processed output audio sample rate). This setting is only used in INT configurations
USB sample rate	-io16 / -io48	to choose between a 16 kHz or 48 kHz rate for the USB rate. Input and output rates must be the same. This setting is only used in UA configurations
Microphone topology	-sqr / -lin	to choose between a "squarecular" or linear microphone array
Control protocol	-i2c / -spi	to choose between I^{2}C and SPI control modes. This setting is only used in INT configurations, the UA configurations always use USB control mode
Audio MCLK	-extmclk	to choose between external or internal MCLK. This setting is only used in INT configurations

Note: Some binaries are provided which have the suffix `-extmclk`. These are intended for use in systems where an external MCLK is provided, and they disable the XVF3800's clock recovery system. These builds are not for use with a Raspberry Pi.

The firmware is installed using the XMOS XTAG4 adapter supplied with the XK-VOICE-SQ66 development kit.

1. Connect an XTAG-4 debug adapter to the setup computer via USB, and connect it to the XK-VOICE-SQ66 development kit using the supplied ribbon cable. The cable should be plugged into XSYS2 connector on the XK-VOICE-SQ66 development kit.

2. Open an XTC tools terminal window on the computer. Verify that the XTAG4 has been correctly connected by running the following command in that window:

```
xflash -l
```

The output from this should be of the following form:

```
Available XMOS Devices
----------------------

  ID    Name          Adapter ID  Devices
  --    ----          ----------  -------
  0     XMOS XTAG-4   ABCDE123    P[0]
```

Note: If the XTAG-4 is not properly connected to the development machine, then `xflash` will report `No Available Devices Found`. If the XK-VOICE-SQ66 development kit is not properly connected to the XTAG-4, then the Devices column will read `None`. For further guidance on the use of the *XTC tools*, see the https://www.xmos.ai/view/Tools-15-Documentation.

---

3. Select the required binary firmware image from the release package, and transfer it to the XK-VOICE-SQ66 development kit using the `xflash` tool.

```
xflash application_xvf3800_intdev[...].xe
```

The XTAG4 adapter can be disconnected from the XK-VOICE-SQ66 development kit after successfully installing the firmware image if desired.

After completing the setup, follow the instructions below to set up the audio path.

2.4 Setting up the Audio

2.4.1 Loudspeaker

To play reference audio into the room, a high quality loudspeaker operating in its linear region is required. Connect the loudspeaker to the LINE OUT port on the XK-VOICE-SQ66 development kit, which accepts a 3.5 mm TRS jack plug connector. It is important for the ideal demonstration that the position, orientation, and volume of the loudspeaker are representative of a real-world system.

To achieve optimal performance:

- Align the front of the loudspeaker with the microphone strip at the top of the XK-VOICE-SQ66 development kit and
- Place the loudspeaker 2-4 cm away from the device

An example layout can be seen pictured in Fig. 2.4.

Fig. 2.4: XVF3800 demo example layout

To calibrate the volume of the loudspeaker for optimal performance, a test file such as the IEEE 269-2010 Male Mono 48 kHz signal can be used.

These test signals can be downloaded from:

```
https://standards.ieee.org/wp-content/uploads/import/download/269-2010_downloads.zip
```

and stored on the computer used for playback (USB host or I2S host). If the Raspberry Pi is connected to the internet the files can be copied with the following:

```
curl --output <FILE>.zip https://standards.ieee.org/wp-content/uploads/import/download/
↪269-2010_downloads.zip
unzip <FILE>.zip
```

The output volume must be changed directly on the loudspeaker or on a connected amplifier, not on the computer. The volume of the track measured at a 1 metre distance from the loudspeaker should be 73 dB$_A$±2 dB$_A$ on average.

2.4.2 Playback and Recording

2.4.2.1 I2S Audio Configuration

Playback and recording through the XVF3800's I^{2}S interface, once the Raspberry Pi has been set up correctly, can be achieved through standard use of the ALSA card `snd_rpi_simple_card, device 0`. For example, from the command line, a 2 channel 32-bit 48 kHz WAV file may be played as reference audio through `aplay` with the following command:

```
aplay -c 2 -f S32_LE -r 48000 -D hw:sndrpisimplecar,0 <filename>
```

Similarly, the returned processed audio from the XVF3800 may be recorded to a file with

```
arecord --mmap -c 2 -d <time> -f S32_LE -r 48000 -D hw:sndrpisimplecar,0 <filename>
```

Alternatively, in a desktop environment on the Raspberry Pi, Audacity ^{TM} may be used to visually play and record. The sample rate must be set to match the XVF3800 in the *Project Rate (Hz)* selection in the bottom left.

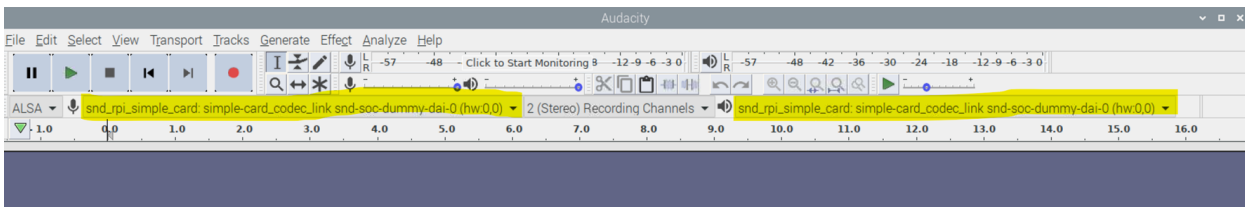The sound card settings must match the ones highlighted in Fig. 2.5:

Fig. 2.5: Audacity ^{TM} example

2.4.2.2 UA Audio Configuration

In the UA configuration, Audacity ^{TM} may be used to visually play and record signals from the device. The sample rate must be set to match the XVF3800 in the *Project Rate (Hz)* selection in the bottom left.

The device should appear as a USB sound card with the default name of `XMOS |project| Voice Processor`. On Audacity both playback and recording devices should be set to be the XVF3800.

With the standard evaluation firmware the USB parameters are:

```
Product_STR: XMOS XVF3800 Voice Processor
Manufacturer_STR:  XMOS
Vendor_ID (VID):  0x20b1
Product_ID (PID): 0x4f00
Serial Number: 00000000
```

These can be modified, along with other USB parameters - see the subsection about *Changing Default Parameter Values*, for details.

2.5 Troubleshooting

2.5.1 Audio Signals

If audio is being played by the host but not heard from the loudspeaker, it is likely that there exists a connection issue between the host and the XK-VOICE-SQ66 development kit board. Ensure that the XK-VOICE-SQ66 development kit board is powered, and the loudspeaker connected and powered.

If audio is heard from the loudspeaker but no input audio is received by the host, ensure that the host is configured to transmit and receive audio at the correct sample rate (either 16 kHz or 48 kHz, depending on the chosen firmware).

If there is still no input from the device, it is possible that the device firmware has stalled; disconnect and reconnect power to the XK-VOICE-SQ66 development kit to reset the device, and then attempt input again.

2.5.2 AEC Convergence

The AEC requires a reference signal be present in order to converge on a room transfer function estimate - this process will take a few seconds after reference audio has begun being provided. If the AEC has not been allowed to converge, the XVF3800 will tend to over-suppress near-end speech in its output to avoid undesirable artefacts being relayed to the far-end. This effect does subside within the first few seconds of use, so if the device is unexpectedly restarted then performance will be reduced momentarily but should restore over time.

If the device has consistently poor acoustic performance, it is likely that the AEC has not converged appropriately; restart the device and repeat the AEC convergence procedure described in the later sections to reset the AEC to a new set of coefficients.

If this does not resolve the issues, it is permissible to lower the loudspeaker volume. If loudspeaker volume is adjusted, or if there is any other change in environment, ensure that the AEC has reconverged before proceeding by playing several seconds of far-end audio again. The AEC will constantly reconverge, so a small change in environment such as a window opening or a change in loudspeaker volume should be automatically adjusted for by the AEC; however, actions that alter the direct delay path significantly, such as moving the loudspeaker, will require the device be reset and the AEC be allowed a few seconds to reconverge from startup.

3 Using the Host Application

The XVF3800 contains a control interface that enables users to configure the operation of the device and to set and read parameter data.

In 2.0 a sample host application, `xvf_host` (Linux, MacOS, Raspberry Pi OS) or `xvf_host.exe` (Windows), is provided which can be used to connect to the control interface on the XVF3800. Please contact XMOS for information on using these tools on other host platforms.

Before using the host application, the host and hardware must be configured as described in *Setting up the hardware*.

3.1 Installing the Host Application

The sample `xvf_host(.exe)` application can be found in the evaluation binaries release package in the sub-folder `host_v<version>/<platform>`. The supported platforms are `linux_x86`, `mac_x86`, `rpi`, `windows`. This whole folder needs to be transferred to the host machine. It can be placed in any convenient location. This directory should contain the following files:

```
.
├── (lib)command_map.(so/dll/dylib) # All platforms
├── libdevice_i2c.so                # RPi only
├── libdevice_spi.so                # RPi only
├── (lib)device_usb.(so/dll/dylib)  # All platforms
├── libusb-1.0.0.dylib              # Mac_x86 only
└── xvf_host(.exe)                  # All platforms
```

To verify the `xvf_host(.exe)` application is installed, change to the directory and run the application as per the example below, setting the appropriate permissions first.

```
chmod +x xvf_host(.exe)
xvf_host(.exe) --help
```

Users may find it convenient to store the host tools in a directory such as `~/bin` and add this to the $PATH environment variable so that the tools can be invoked from any directory, for example with the shell command

```
PATH=~/bin:$PATH
```

or use "Edit system environment variables" GUI on Windows.

After installing the `xvf_host(.exe)` application, only Windows requires some extra steps to install the correct USB driver. The following instructions must be followed:

1. Connect the XVF3800 board to the host PC using a USB cable.

2. If some USB driver has been installed for the *XMOS DFU Factory (Interface 4)* as described in *DFU application setup*, the device must be disabled by using *Device Manager*:

   a. Open *Device Manager* and find the *XMOS DFU Factory (Interface 4)*, this is usually located under the *USB Serial Bus Devices*.

   b. Right click on the interface and select *Disable device* as shown below:
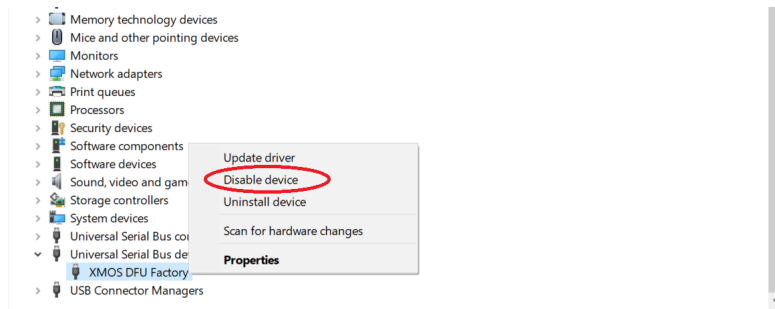
Fig. 3.1: Disable the control device in Device Control

c. Click *Yes* if the message below appears:
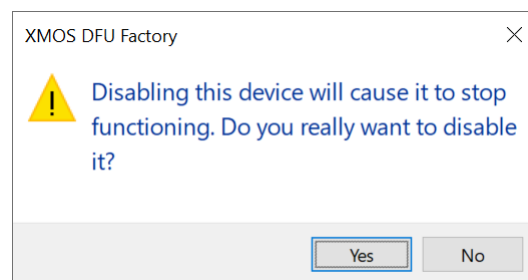
Fig. 3.2: Accept warning message

Note: The driver can be re-enabled again by selecting *Enable device* after a right-click on the device in *Device Manager*

3. Download the third-party installation tool *Zadig* from https://zadig.akeo.ie/.

4. Open *Zadig* and select *XMOS Control (Interface 3)* from the list of devices. If the device is not present, ensure *Options -> List All Devices* is checked.

5. Select *libusb-win32* from the list of drivers.

6. Click the *Install Driver* button and wait for the installation to complete.
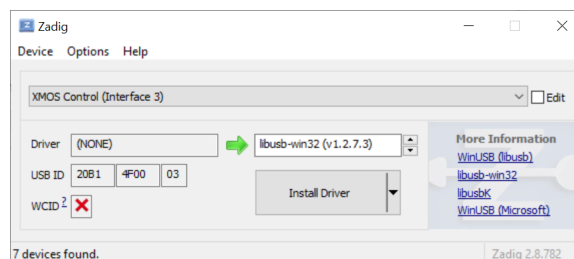
Fig. 3.3: Selecting the *libusb-win32* driver in Zadig for the Control Interface

3.2   Connecting to the XVF3800 Device

To use the host application, login to the host machine, either directly, via a VNC connection or `ssh` and to open a terminal command line.

Change to the directory containing `xvf_host(.exe)`. If the host tools have been added to the path as above this step is not needed.

The `xvf_host(.exe)` device control application is run from the command line.

To check connection to the XVF3800, any command can be given; for example, the command

```
xvf_host(.exe) --use <protocol> VERSION
```

where `<protocol>` can be `i2c`, `spi` or `usb` depending on the interface used in the specific firmware. The default control protocol is USB.

This command should return "2 0 0".

3.3   xvf_host Command Syntax

The general syntax of the `xvf_host(.exe)` command is

```
xvf_host(.exe) [ command | option ]
               [ -u <protocol>] [ -cmp <path> ] [ -br ] [ command | option ]
```

More documentation on the available options in the use of the host application are found with

```
xvf_host(.exe) --help
```

A full list of control commands may be found using:

```
xvf_host(.exe) --list-commands
```

These commands are also listed in an Appendix of this User Guide.

It is possible to read all the control parameter settings from the XVF3800 using the following option:

```
xvf_host(.exe) --dump-params
```

To support scripted set up of the XVF3800 it is possible to save the list of commands in a text file (.txt) which can be executed using:

```
xvf_host(.exe) --execute-command-list <command_file>.txt
```

Further options for saving and loading parameter sets can found by using the `--help` option or in the section of the User Guide on tuning the device where their usage is described.

3.4   Microphone orientation

The XVF3800 supports arbitrary microphone geometries which can be specified in configuration files. Two default configurations are included in the release package that are aligned to the geometries supported on the XK-VOICE-SQ66 development kit.

- The linear configuration (-lin) comprises 4 microphones in a linear array, spaced 33mm apart, as shown below:
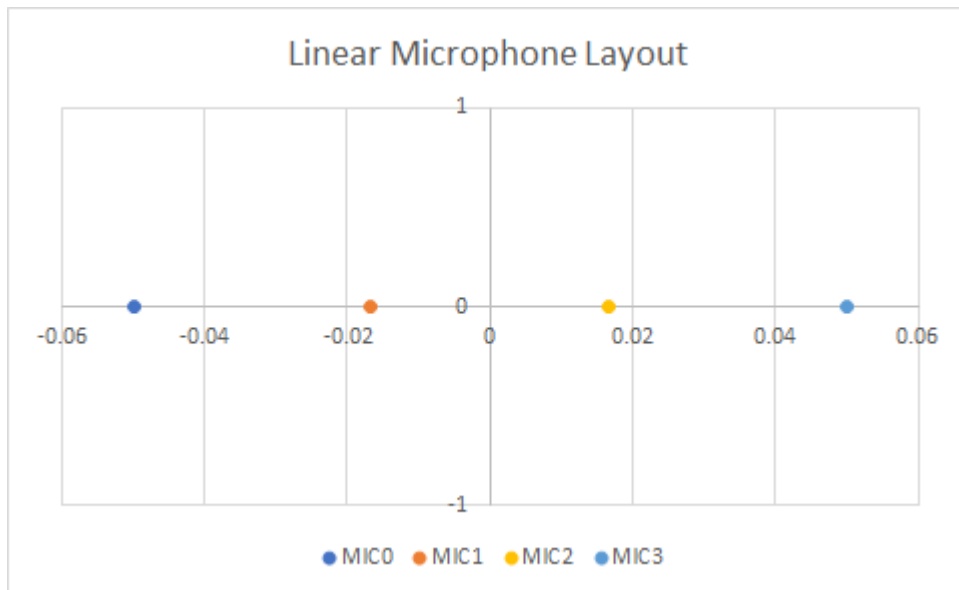
- The square configuration (-sqr) uses a 4 microphone array with a 66mm distance along each side as shown below.
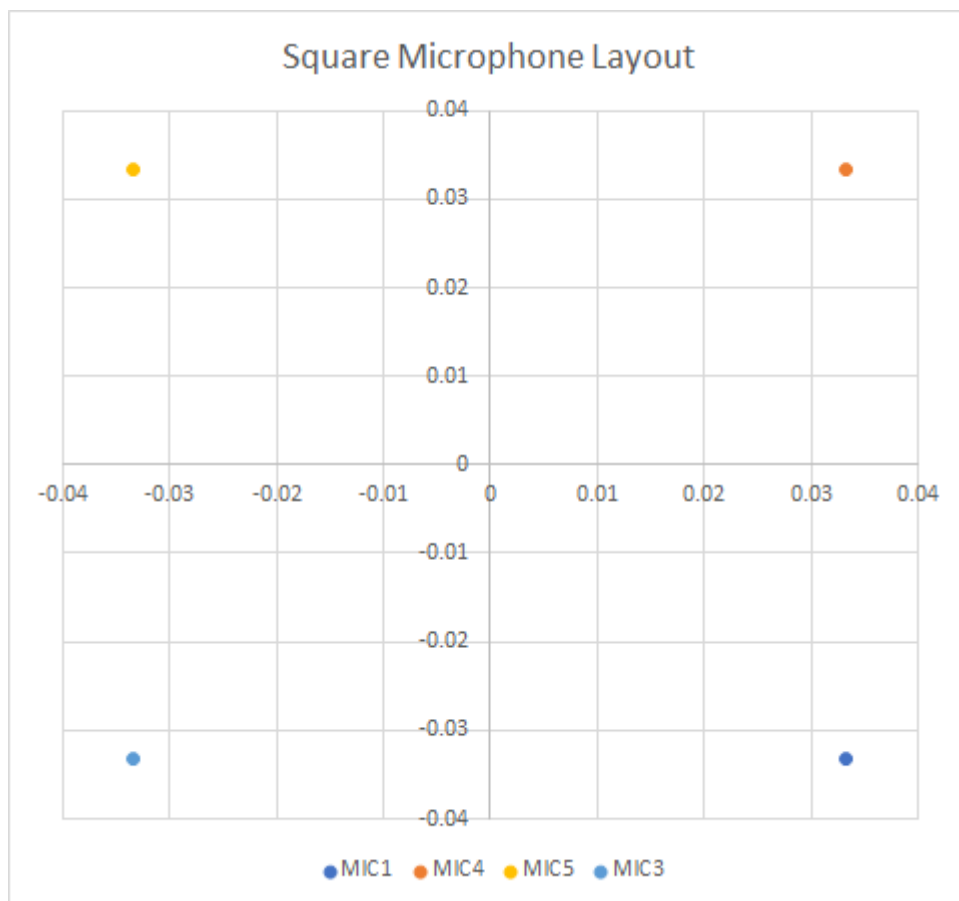
Note: These configurations are selected by a multiplexer on the XK-VOICE-SQ66 development kit when the device boots. They have to be configured as part of the firmware build and cannot be changed in operation.

The microphone numbers in the diagrams above correspond to the labels on the XK-VOICE-SQ66 development kit board. In the firmware configuration these microphones are mapped to 4 microphone indexes:

MIC[0..3] which are used in the firmware and when using `xvf_host`. The mapping of the logical indexes and the physical microphones on the XK-VOICE-SQ66 development kit is shown in the table below:

Table 3.1: Logical Microphone Index to physical microphone mapping

Logical	Physical Microphone	
Microphone Index	Linear Config	Square Config
MIC0	MIC0	MIC1
MIC1	MIC1	MIC4
MIC2	MIC2	MIC5
MIC3	MIC3	MIC3

3.4.1 Beam forming subsystem and Direction of Arrival indicator

As described in XVF3800 datasheet, the system uses a set of beams to focus on speakers and reduce unwanted sounds and reverberation in the output signal. The XVF3800 uses a free running beam that scans the environment, identifies likely speakers and switches one of the two focused beams to that direction.

In normal operation the audio pipeline automatically selects the best signal to output.

It is possible to read back the direction that the beams are currently pointing. This is done with the `xvf_host` command AEC_AZIMUTH_VALUES. The output of the command contains 4 values:

- Focused beam 1
- Focused beam 2
- Free running beam
- Auto selected beam

Each value is the azimuth angle of the corresponding beam, provided in both radians and degrees. The coordinate system used depends on the hardware configuration. The diagrams below show the azimuth angles relative to the XK-VOICE-SQ66 development kit in the two default configurations.
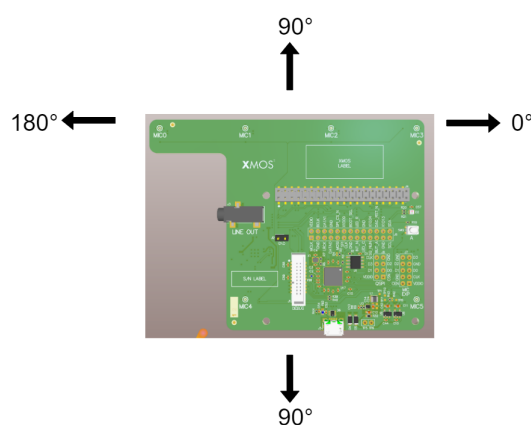
Fig. 3.4: Azimuth angle in linear configuration (note - 0 to 180 degrees only)
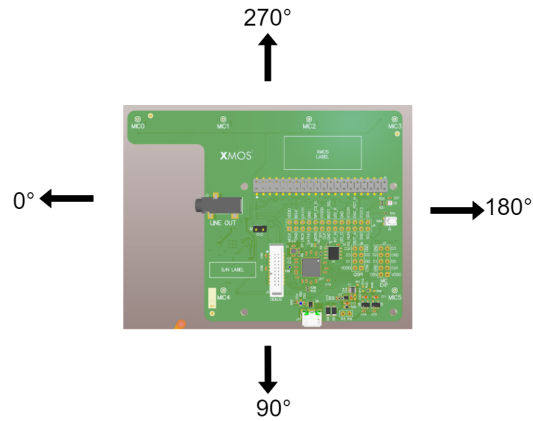
Fig. 3.5: Azimuth angle in square configuration

3.4.2   Using azimuth data For Direction of Arrival indication

The auto selection algorithm will switch between beams rapidly in some circumstances. The two focused beams update relatively slowly, but the free running beam is designed to be sensitive so that it can rapidly pick up the speech signal for a new talker entering the soundscape. As a result it can also pick up any noise signals present.

To provide maximum flexibility to system designers, the XVF3800 only provides raw azimuth data. In most use cases this data will require post processing - e.g. to only track the slow focused beams or to add a voice activity detector to indicate when the direction of arrival data is valid.

3.5   Example Uses

The `xvf_host` tool allows the configuration of the XK-VOICE-SQ66 development kit to be changed during operations. The following examples illustrate some common operations.

Note: All examples below will target UNIX-based platforms

3.5.1   Output Selection

By default, the left (first) channel of the device's output is the processed output from the XVF3800's AEC and beamforming stage, while the right (second) channel is the raw input from one of the microphones after amplification. This selection provides a good comparison between the raw and processed audio. The selected outputs may be changed by using the *AUDIO_MGR_OP_L* and *AUDIO_MGR_OP_R* commands. These commands each take two integers defining the mux routing settings, described as a pair of (category, source) values.

The available categories and sources are as detailed in Table 3.2.

Table 3.2: Audio manager mux options

Category	Sources
0: Silence	0: Silence
1: Raw microphone data - before amplification	0,1,2,3: Specific microphones accessed by index, no system delay applied
2: Unpacked microphone data	0,1,2,3: Unpacked microphone signals. If using packed input, access packed microphone data though this category. This data is undefined when not using packed input.
3: Amplified microphone data with system delay	0,1,2,3: Specific microphones accessed by index. This category provides the microphone signal passed to the SHF logical cores for processing. Source 2 is the default setting for the right channel output.
4: Far end (reference) data	0: Far end data received over I^2S, post sample rate conversion to 16 kHz if required
5: Far end (reference) data with system delay	0: Far end data received over I^2S, post sample rate conversion to 16 kHz if required, and with system delay applied
6: Processed data	0,1: Slow-moving post-processed beamformed outputs, 2: Fast-moving post-processed beamformed output, 3: The "auto-select" beam; chooses the best of the previous three beams as an output, recommended option for selecting the beamformed outputs
7: AEC residual data	0,1,2,3: AEC residuals for the specified beam
8: User chosen channels	0,1: These currently copy the auto-select beam (category 6, source 3) and are the default setting for the left channel output.
9: Post SHF DSP channels	0,1,2,3: All output channels from user post SHF DSP
10: Far end at native rate	0,1,2,3,4,5: Data passed from I^2S logical core to Audio Manager logical core. All sources carry useful data if the external interface rate is 48 kHz. Only sources 0 and 1 carry useful data if the external interface rate is 16 kHz. See the Data Plane Detailed Design section in the Programming Guide for information on the interface between these two logical cores.
11: Amplified microphone data before system delay	0,1,2,3: Specific microphones accessed by index.
12: Amplified far end (reference) with system delay	0: Far end data received over I^2S, post sample rate conversion to 16 kHz if required, and with a configurable fixed gain and system delay applied. This category provides the reference signal passed to the SHF logical cores for processing.

Fig. 3.6 shows the available categories and sources. Inputs to the category multiplexer that have no preliminary multiplexer only support a single source.
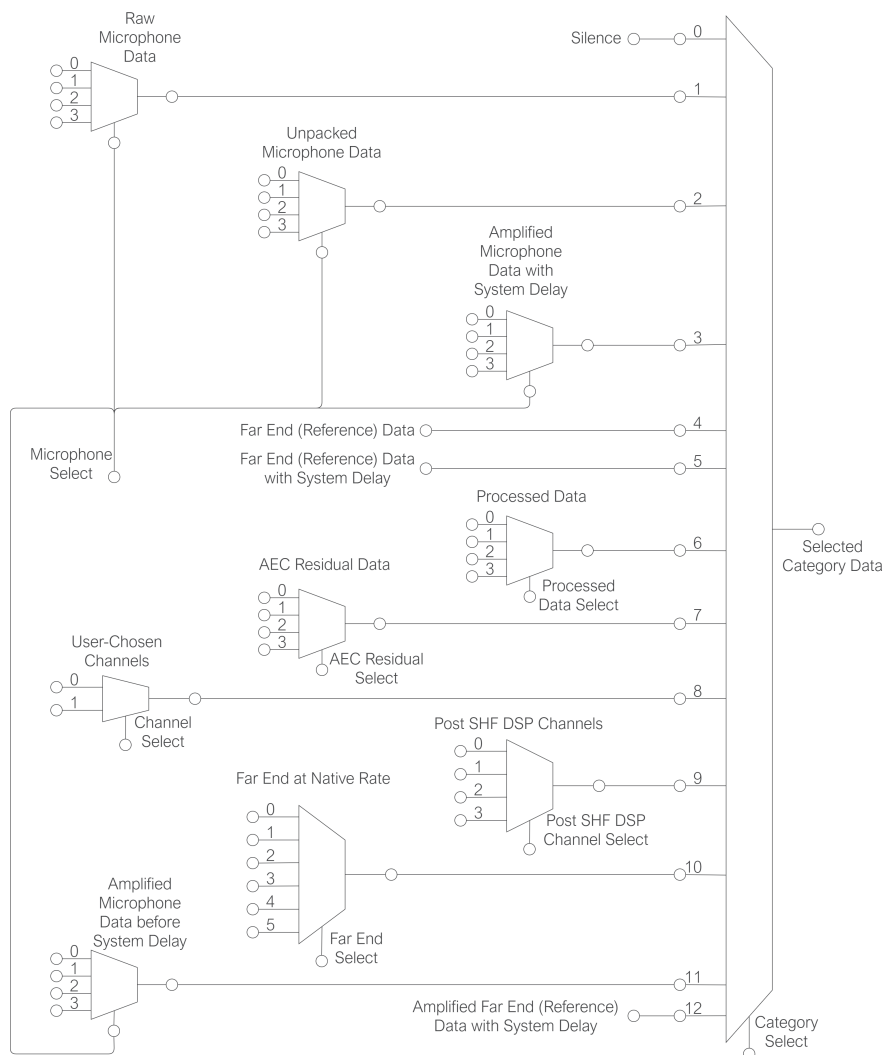
Fig. 3.6: Output Selection Multiplexers

For example, to set the left output to the 4th raw microphone signal (without gain applied), issue the command

```
xvf_host AUDIO_MGR_OP_L 1 3
```

This will set the left channel to output the 4th (0-indexed) microphone signal of the 4 present. To reset this channel back to its default value, issue

```
xvf_host AUDIO_MGR_OP_L 8 0
```

to set the channel to the postprocessed auto-selected output beam.

Similarly, the right channel may be set to any desired category/source; to reset to its default value, issue

```
xvf_host AUDIO_MGR_OP_R 3 2
```

3.5.2   Setting an Output Pin

The `xvf_host` can be used to configure the General Purpose Outputs on the XVF3800.

To turn on the LED on the XK-VOICE-SQ66 development kit issue the following commands

```
xvf_host gpo_port_pin_index 0 6
xvf_host gpo_pin_pwm_duty 100
```

and to turn it off use

```
xvf_host gpo_pin_pwm_duty 0
```

Note: The `gpo_port_pin_index` selects a port and subsequent `gpo_pin_xxx` commands only act on that pin.

---

4 Tuning the Application

The measured performance of the XVF3800 depends very heavily on the electrical and acoustic environment of the end product that it is incorporated into. In order to achieve optimal performance, including the ability to pass product certification tests, it is necessary to perform a configuration and tuning process to adapt the firmware to the end product's form factor and hardware design.

The majority of this configuration is intended to ensure optimal performance of the XVF3800 audio pipeline, including the behaviour of the Adaptive Echo Canceller (AEC).

The full set of configurable parameters for the XVF3800 is given in the *Appendix*.

This chapter makes extensive use of the `xvf_host` application to control configuration parameters at run-time. For further documentation on this utility, please see the section *Using the Host Application*. Throughout this document, the `-u [i2c|spi|usb]` parameter to this utility will be omitted for brevity.

To facilitate the tuning process a set of software tools are also supplied to process measurements. These tools are provided as python programs and can be found with the host application in the XVF3800 source release package.

4.1 System Preparation

4.1.1 Prerequisites

There are a number of prerequisites that should be met in order to facilitate the tuning process:

- It must be possible to both play arbitrary reference input through the XVF3800 over I^{2}S and to record the device's output.

- It must also be possible to access the control interface on the XVF3800, either through I^{2}C or SPI as desired.

- Create a block diagram of the whole system, showing audio path from input through to output and including the XVF3800. This can be used to understand how to optimise and control the performance of the overall product. Ensure that the path from the reference input through to the loudspeaker and from the microphones to the XVF3800, including any gain, EQ, compression, filtering, and limiting applied, are illustrated. Ensure also that the points where control is available over these parameters (and, more importantly, where it is not) is fully understood. Fig. 4.1 provides a block diagram of the audio functions in the XVF3800 to aid in the preparation of this system diagram.

- Ensure a good understanding of the coherence between the individual microphones; see the discussion of *microphone coherence* in the acoustic guidelines section for details and requirements on this.

- Further, ensure a good understanding of the delay between the microphones and the reference signal input; see *section on system delay* for details, requirements, and terminology surrounding this.
  - This delay should remain constant while the device is running. Any inconsistency in this delay will result in severely degraded algorithmic performance.
  - If this delay should change between device reboots, for example due to any front-end processor used to receive the far-end signal, it is important that the device remain causal.

- Care should be taken that samples not be dropped between the device's reference audio input and the XVF3800.
  - In addition, ensure that any clocking jitter on the interface that carries the reference signal, such as I^{2}S or a USB interface, is minimised.

- Access to the IEEE 269-2010 reference signals is useful for representative clear speech signals. Alternative speech signals may be found from the ITU, in particular the files associated with Recommendation P.501, which at time of writing may be acquired from the webpage for this ITU recommendation. Files from these two sets will be referred to in this document by filename.

- All the tuning tools and the audio tracks are included in the release package in the folders below:

  - `/sources/app_xvf3800/acoustic_design/`

  - `/sources/app_xvf3800/nl_model_gen/`

  - `/sources/app_xvf3800/tuning/`
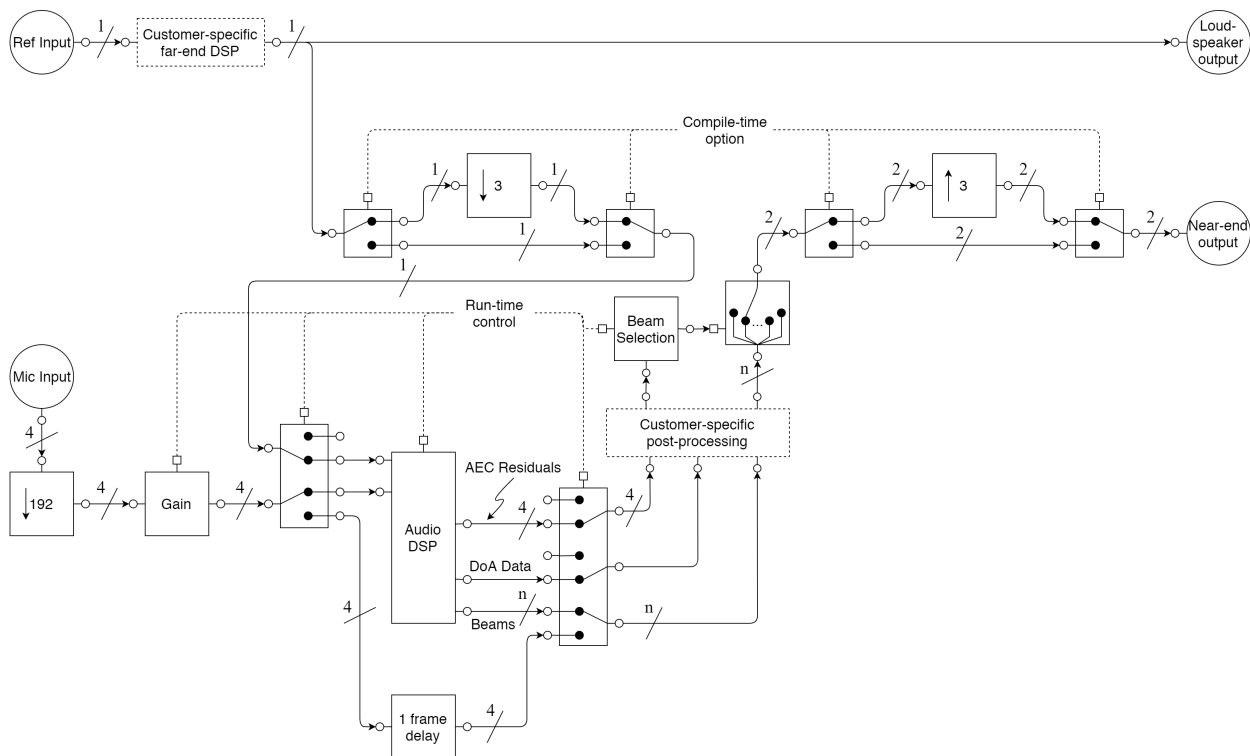
Fig. 4.1: XVF3800 audio block diagram

- Install or update the following tools and python modules to enable the tuning scripts to run:

  - Audacity

  - Python3

  - python modules: numpy, matplotlib, scipy, asyncssh (installed via pip)

  - sox

4.1.2 Initial Parameter Setting

There are a selection of parameters that should be chosen before the tuning process starts, and will not be modified during the provided tuning process:

- `AEC_HPFONOFF`: This sets a high-pass filter (HPF) on the microphone signals as they enter the processing block; this takes the form of a 4th order Butterworth filter, and therefore has a -80 dB per decade rolloff. The corner frequency (-3 dB point) for this HPF may be set to 70 Hz, 125 Hz, 150 Hz, 180 Hz, or the filter may be disabled.

- **AEC_FAR_EXTGAIN**: This parameter informs the audio pipeline how much external gain has been applied to the AEC reference signal. The value that this parameter should take is coupled to the volume control of the device; if the device attenuates the signal by e.g. -6 dB, this value should be set to -6.

- **AEC_AECSILENCELEVEL**: This sets a power threshold for signal detection in the AEC. If there is known e.g. ADC induced noise in the reference audio signal line, this parameter may be set to avoid the AEC adapting to this noise.

- **PP_LIMITONOFF** and **PP_LIMITPLIMIT**: A power limiter may be inserted in line with the processed audio outputs from the audio pipeline using **PP_LIMITONOFF**. The power threshold used may be set with the **PP_LIMITPLIMIT** command. If the output energy is predicted to exceed **PP_LIMITPLIMIT**, compression is applied to the outputs to avoid this.

- **PP_DTSENSITIVE**: **PP_DTSENSITIVE** allows some control over the balance struck between double-talk performance and echo suppression, including the use of an optional near-end speech detector. This is summarised in Fig. 4.2; as echo suppression increases, double-talk performance will tend to decrease as more near-end is suppressed.
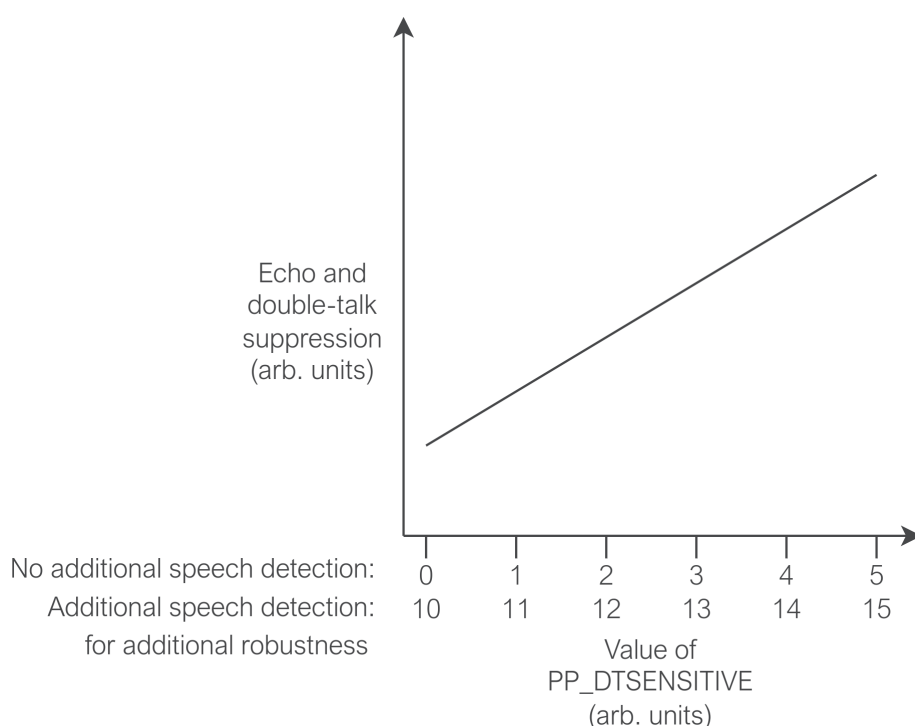
Fig. 4.2: Illustration of relationship between **PP_DTSENSITIVE** value and echo suppression

4.1.3 Initial Tests

The first step in tuning the product is to ensure that the send, receive, and loopback paths through the XVF3800 are electrically stable and that the XVF3800 has a stable control interface.

4.1.3.1 Input Path

This test will attempt to verify that a signal injected into the device through the device's intended input path successfully reaches the XVF3800. This tests path 1 in Fig. 4.3. If possible, inject an test signal (such as white noise) through the device's reference audio input path and monitor the signal path immediately prior to the XVF3800. Consider disabling the device's loudspeaker for the duration of this test if the test signal chosen would cause auditory discomfort. Verify that the test signal is observed.
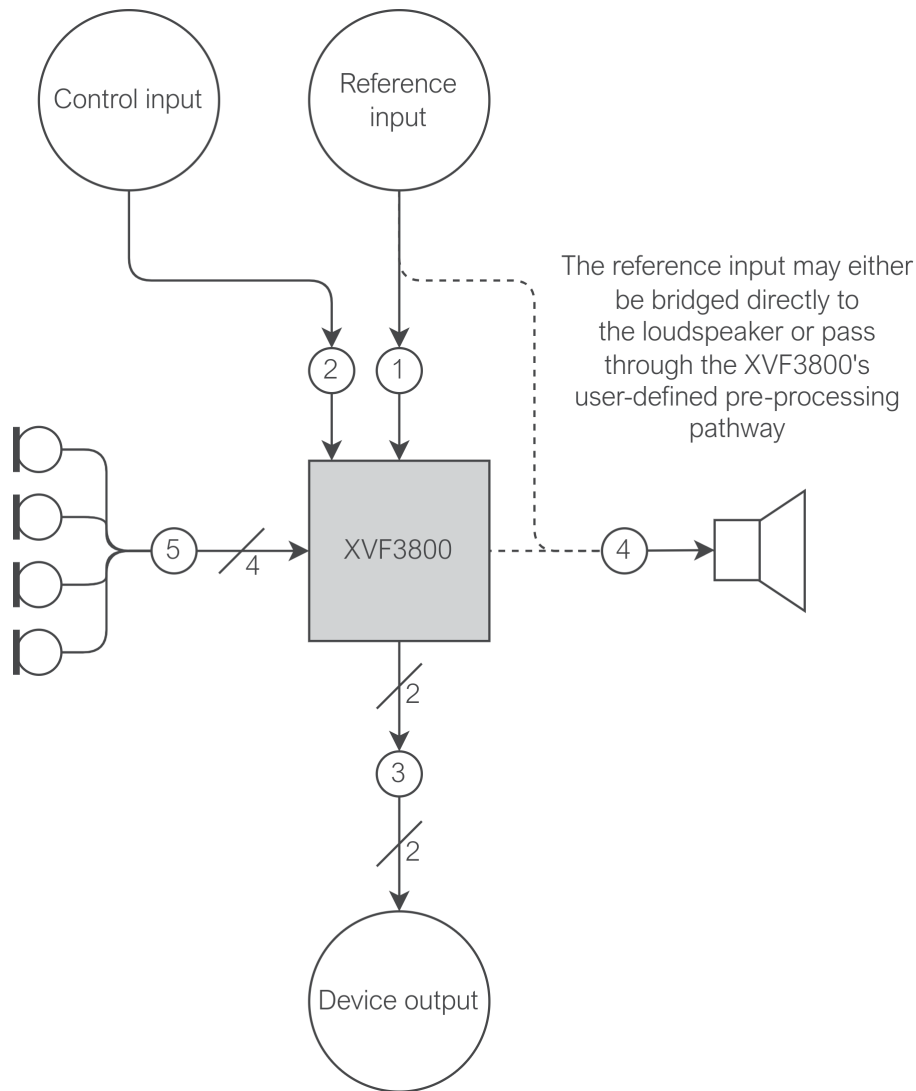
Fig. 4.3: Top-level schematic showing the 5 input and output paths of the XVF3800

In Fig. 4.3, numbered points correspond to the suggested order of testing.

Note: If direct monitoring of the signal path immediately prior to the XVF3800 is not feasible, it is permissible to skip this test; its function is implied in later tests.

4.1.3.2 Control Path

This test will attempt to verify that the XVF3800 has a stable control interface. This tests path 2 in Fig. 4.3. Following the guidelines in *Using the Host Application*, issue:

```
xvf_host VERSION
```

and ensure that the device returns "2 0 0".

4.1.3.3 Output Path

This test will attempt to verify that a signal injected into the XVF3800 is output faithfully from the XVF3800 and successfully output by the device. This tests paths 1 and 3 in Fig. 4.3. Set up the XVF3800's output mux as follows to loop back any I²S data received:

```
xvf_host AUDIO_MGR_OP_UPSAMPLE 0 0
xvf_host AUDIO_MGR_OP_ALL 10 0 10 2 10 4 10 1 10 3 10 5
```

Note: As the signals produced here are by definition at the I²S sample rate, they do not require the use of the upsampler in the case of a 48 kHz I²S bus, and therefore we explicitly unset the `AUDIO_MGR_OP_UPSAMPLE` flags (set to 1 by default in a 48 kHz configuration). If using a 48 kHz I²S bus, be sure that `AUDIO_MGR_OP_UPSAMPLE` is reset appropriately to accommodate signals that are generated at 16 kHz, including the processed output signals.

Inject a test signal (such as white noise) through the device's reference audio input path and monitor the signal on the device's communications output. Verify that the output matches the input signal. This should have a fixed delay, but should otherwise be the raw I²S data, after any customer-specific pre-processing DSP has been applied. Consider disabling the device's loudspeaker for the duration of this test if the test signal chosen would cause auditory discomfort.

4.1.3.4 Speaker Operation

It is advised that the linearity, stability (even operation over the desired frequency range), output level, and total harmonic distortion (THD) be characterised for the loudspeaker(s) in use in the product. Play a test file, such as an IEEE Reference file, through the loudspeaker and observe the output level. The loudspeaker level should be adjusted such that it meets the desired output level target. This tests path 4 in Fig. 4.3.

Note: The desired loudspeaker output level is usually specified by product certification requirements such as those constructed by Amazon, Microsoft, or Zoom. Refer to your desired certification requirements for appropriate targets for this test.

With the loudspeaker at an appropriate level, observe that there is not audible distortion or nonlinearity present in the speech signal. If desired, make a quantitative measurement of THD to ensure that the loudspeaker is operating as intended. Correct operation of the loudspeaker is essential to the tuning process. Operating the loudspeaker and associated amplifier within their linear region is highly important for the tuning process and for optimal algorithmic performance.

4.1.3.5 Microphone Operation

Ensure that the microphone assignment is as expected and that they sound natural and artefact-free. This tests paths 3 and 5 in Fig. 4.3.

Set up the XVF3800's output mux as follows to output raw data for microphones 0 and 1:

```
xvf_host AUDIO_MGR_OP_L 1 0
xvf_host AUDIO_MGR_OP_R 1 1
```

and set up as follows to output raw data for microphones 2 and 3:

```
xvf_host AUDIO_MGR_OP_L 1 2
xvf_host AUDIO_MGR_OP_R 1 3
```

Note: As the microphone signals are decimated to 16 kHz within the XVF3800's audio manager, the microphone signals require upsampling on a 48 kHz bus - ensure that the AUDIO_MGR_OP_UPSAMPLE flags have been reset after previous testing if using a 48 kHz XVF3800 configuration.

Ensure that each microphone is assigned as expected; this can be achieved by e.g. clicking near or tapping each microphone in turn to ensure that the signal is routed to the expected output. If the microphone assignment is not as expected, then the microphone geometry may be incorrect and therefore Direction of Arrival (DoA) information may be incorrect.

Record some near-end signal (such as speech) and analyse the result for undesirable artefacts, such as noise, distortion, or interference. Ensure that speech through each microphone sounds clear and natural. Verify that each microphone is similar in level, for example by examining a power spectral density plot (PSD) of a known near-end source and observing that each microphone signal has a similar total power.

4.2 Tuning the XVF3800 Parameters

This section will walk through a typical tuning process, step by step. It is advised that, when appropriate values for each tuning parameter are determined, the device firmware is rebuilt with these values as default and the device is reflashed, This process is described in the section *Building the application*. It is recommended that the device be restarted at the start of each of these tuning steps.

4.2.1 Reference Gain

The `AUDIO_MGR_REF_GAIN` parameter is provided to control a gain block placed in the reference audio path after the customer-specific pre-processing DSP stage. The reference audio should be amplified such that any peak amplitude losses through the input path (such as attenuation or filtering prior to the XVF3800 or in the customer-specific pre-processing DSP stage) are accounted for. This gain is applied within the audio manager internal to the XVF3800, and therefore does not have an impact on the signal sent to the loudspeaker.

Set up the XVF3800's output mux as follows to output pre- and post-gain data for the reference input:

```
xvf_host AUDIO_MGR_OP_L 4 0
xvf_host AUDIO_MGR_OP_R 12 0
```

This will set the left output as the pre-gain reference input, and the right output as the post-gain reference input. With default device configuration, these should be the same.

Depending on the device sample rate, copy from `/sources/app_xvf3800/tuning/` the track `white_noise_0dbfs_2ch_16khz.wav` or `white_noise_0dbfs_2ch_48khz.wav`. Inject the white noise signal into the device's reference input and verify that the reference input observed by the XVF3800 is the same level, i.e. with a peak of 0 dB$_{FS}$. If this is not the case, tune `AUDIO_MGR_REF_GAIN` such that the post-gain reference input has the largest value possible, up to 0 dB$_{FS}$.

Note: White noise is chosen in this example as it contains equal energy in all frequency bands. This is important in cases where e.g. a filter is applied to the reference signal before the XVF3800 or in the customer-specific pre-processing DSP block. In these cases, a single tone may be attenuated more than other tones, and tuning to this specific frequency may lead the device to clip at other frequencies. If no such filter is applied, a tone (such as the tracks `sine_1khz_0dbfs_2ch_16khz.wav` or `sine_1khz_0dbfs_2ch_48khz.wav`) may be chosen instead, which has a more predictable peak amplitude in a shorter timeframe.

Note: It is very important that the reference input can never digitally clip. If this is a risk, it is permissible to leave some headroom in this parameter of approximately 1 - 2 dB.

4.2.2 Microphone Gain

Similarly, the `AUDIO_MGR_MIC_GAIN` parameter is provided to control a gain block placed in the input path from the microphones. The same gain is applied to all four microphones. To tune `AUDIO_MGR_MIC_GAIN`, set the left output as a selected microphone post-gain - for example, microphone 0 - and the right input as the reference audio post gain:

```
xvf_host AUDIO_MGR_OP_L 3 0
xvf_host AUDIO_MGR_OP_R 12 0
```

Depending on the device sample rate, copy from `/sources/app_xvf3800/tuning/` the track `white_noise_0dbfs_2ch_16khz.wav` or `white_noise_0dbfs_2ch_48khz.wav`. Inject the noise signal into the device's reference input and observe the relationship between the post-gain reference signal and the post-gain microphone signal. Tune `AUDIO_MGR_MIC_GAIN` such that the microphone signal has a peak amplitude 6 dB below the reference signal. Observe the other 3 microphone channels and ensure that none exceed 6 dB below the reference signal.

Note: If the microphone signal becomes louder than 6 dB below the reference signal, the AEC may converge to coefficients in the frequency domain greater than 0 dB. This has a significantly negative effect on algorithmic performance, and may lead to instability.

Consider rotating the device, placing it near walls or corners, placing objects in front of the microphones, or exercising other realistic use-cases. Ensure that in each of these cases the post-gain microphone signal does not exceed 6 dB below the reference signal.

4.2.3 System Delay

With an appropriate gain structure, the next step in the tuning process is to ensure that the product is *causal*; that is to say, that an event played in the reference audio stream and over the loudspeaker is received by XVF3800 reference input an appropriate amount of time (in samples) before the coupled signal returns through the microphone path. This is very important; if the system is acausal (a signal played into the room in the reference audio stream is received in the microphone inputs before it is received in the reference input) then effective echo cancellation cannot be achieved. By the same token, the microphones should not be overly delayed compared to the reference input; each coefficient in the AEC corresponds to a sample in the time domain, and so if the microphone signal is overly delayed, fewer AEC coefficients will be of use and the overall behaviour of the AEC will be less optimal.

Expanding on the top-level diagram featured in Fig. 4.3, a more realistic understanding of the main two paths for the reference signal to take can be seen in Fig. 4.4. If the reference signal takes path A pictured, where it is passed through the XVF3800 before it is then sent to the loudspeaker, then it is highly unlikely that the device will become acausal. If instead the signal is sent via path B pictured, where the reference input is passed to the loudspeaker assembly prior to sending on to the XVF3800, an arbitrary reference path delay has the potential to push the device into *acausality* if it exceeds the echo path delay between the loudspeaker and the microphones.
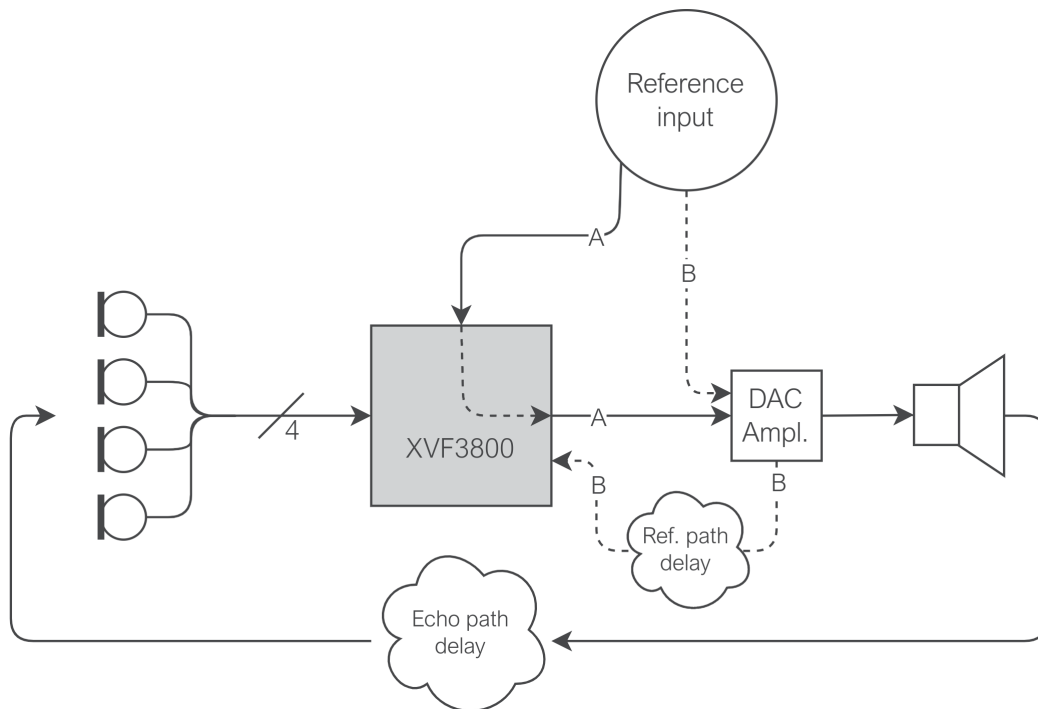
Fig. 4.4: A more detailed representation of the reference input path for the XVF3800, showing both path A where the signal is routed through the XVF3800 (and any customer-specific pre-processing) before sending to the loudspeaker, and path B where the signal is routed via the loudspeaker before it reaches the XVF3800.

In an ideal system, the delay between the reference and microphone signals should be at or less than 40 samples. The `AUDIO_MGR_SYS_DELAY` parameter allows a configurable delay to be applied to either the microphone signal or the reference signal to achieve this 40 sample difference.

A positive value for this parameter, measured in number of samples, sets a delay on the reference signal; if the delay between microphones and reference is too large, setting this value as positive will reduce this difference. A negative value sets a delay on the microphone signals. Setting this delay to a negative value is the recommended method to correct acausality in the device. Note that this will naturally increase the overall delay from input to output through the device.

To estimate the current causality of the system, use the *mic_ref_correlate.py* script provided. To obtain the required signals for this script, set the output mux as follows:

```
xvf_host AUDIO_MGR_OP_L 3 <microphone number>
xvf_host AUDIO_MGR_OP_R 5 0
```

Note: Causality must be checked for all four microphones, as each of the microphones may have a different echo path delay. Calculate correlation between each microphone and the reference in turn.

Depending on the device sample rate, copy from `/sources/app_xvf3800/tuning/` the track `silence_white_noise_0dbfs_silence_2ch_16khz.wav` or `silence_white_noise_0dbfs_silence_2ch_48khz.wav`; these tracks contain 5s of silence, followed by 10s of 0 dB$_{FS}$ white noise, followed by 5s of silence. Pass this through the reference input and record the device output in your chosen audio tools, e.g. Audacity. Save the result as a 2 channel WAV file with the left channel (the post-delay post-gain microphone) as channel 0 and the right channel (the looped-back post-delay post-gain reference signal) as channel 1. Use this as the input to the script:

```
python3 mic_ref_correlate.py [input wav file].wav
```

A diagram similar to Fig. 4.5 should be generated.
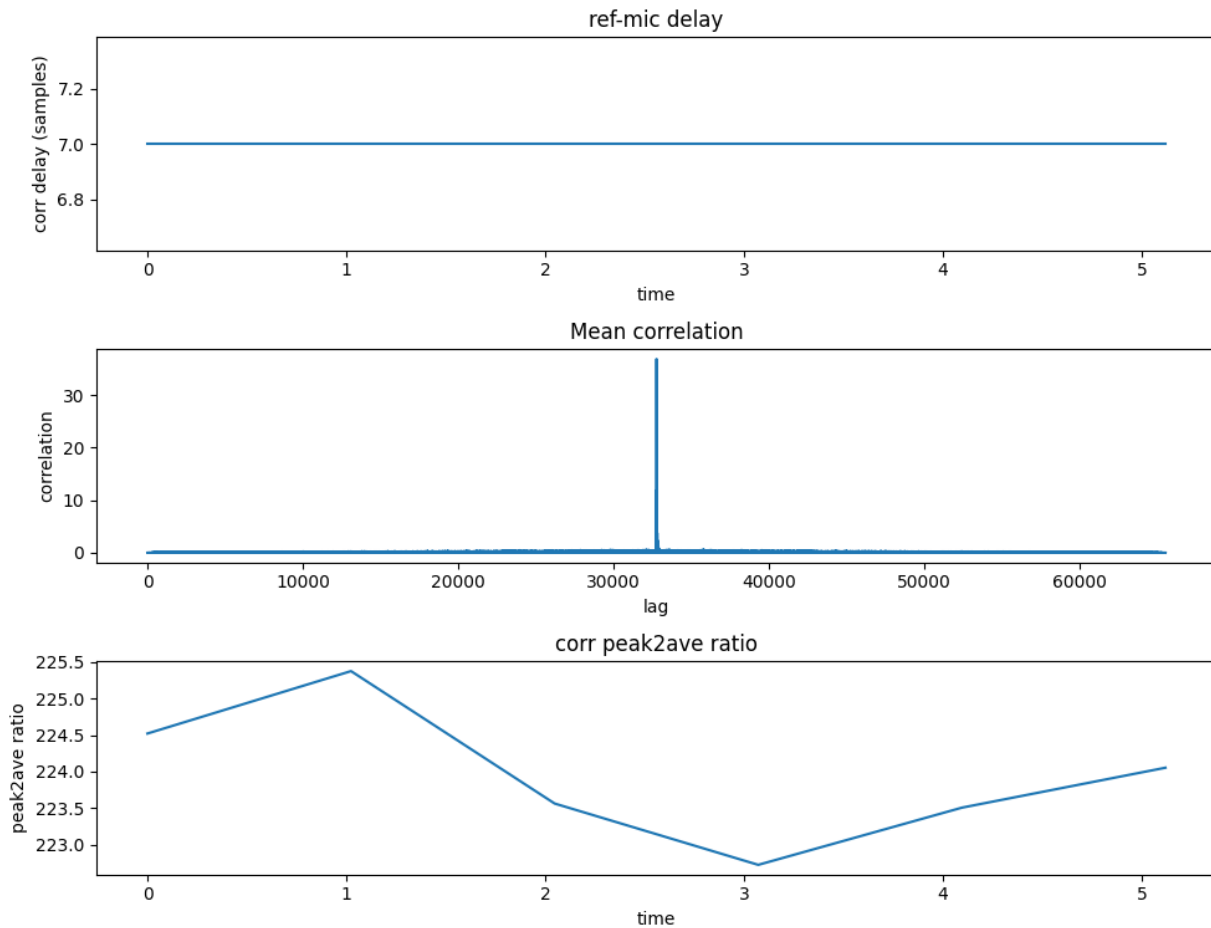
Fig. 4.5: Screenshot showing example output of `mic_ref_correlate.py`

Fig. 4.5 shows a 7 sample delay between microphones and the reference signal. This system is causal, but only just. Setting `AUDIO_MGR_SYS_DELAY` to around 30 will bring the system to the recommended headroom.

This procedure may be repeated after the `AUDIO_MGR_SYS_DELAY` parameter has been set to verify that the system remains causal and has the desired (less than 40 samples) delay between the reference and microphone inputs. Ensure that the device is causal for all four microphones.

4.2.4 AEC Operation

To verify the AEC's operation, play through the reference input a representative test sample, such as `IEEE_269-2010_Male_mono_48_kHz.wav`.

Allow the AEC to converge. The convergence of the AEC may be monitored by use of the `AEC_AECCONVERGED` parameter. This is a read-only parameter. Issuing:

```
xvf_host AEC_AECCONVERGED
```

will present the return value as:

```
AEC_AECCONVERGED [0|1]
```

If the returned value is 1, the AEC has converged.

Note: Once this value is set to 1 internally, it is never reset, even if a significant path change or other circum-

stance forces a significant change in the AEC.

When the AEC reaches convergence (which is expected to take less than 30 seconds), read the AEC coefficients from the device:

```
xvf_host (-gf | --get-aec-filter) [filename.bin]
```

This operation generates 4 separate files <filename.bin>.f0.m[0..3], one per AEC channel. These four files can be analysed in one operation with the *read_aec_filter.py* script provided:

```
python3 read_aec_filter.py [filename.bin]
```

This should generate a plot as shown Fig. 4.6.
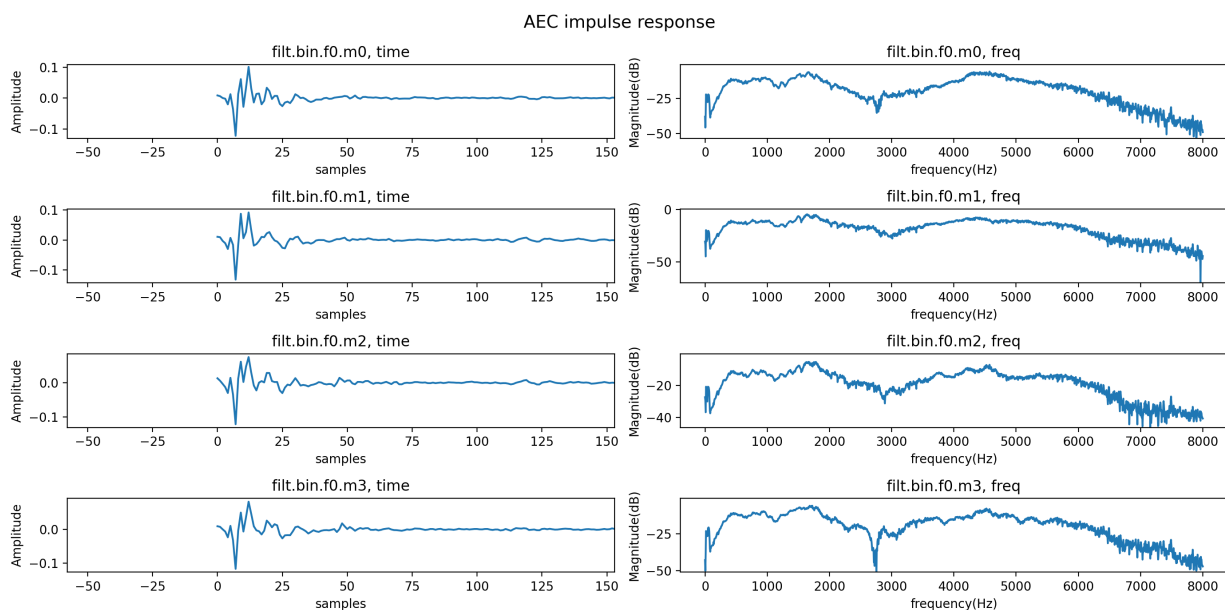
Fig. 4.6: Screenshot showing example output of `read_aec_filter.py`

It will also print in the console the value of the peak coefficient in the frequency domain. Ensure that this is below 0 dB for all four microphones. If it is not, reduce `AUDIO_MGR_MIC_GAIN` to satisfy this.

Observe the period between 0 and 100 samples in the time domain. There should be a strong first peak, as shown in Fig. 4.6. The location of this peak in the time domain should be the same as the previously observed delay between each microphone and the reference input. If this value is significantly above 40 samples, increase `AUDIO_MGR_SYS_DELAY` to reduce this. If the time domain response starts with a strong peak at the first sample, this could be an indication that your system is acausal - reduce `AUDIO_MGR_SYS_DELAY` to attempt to bring the full time domain response into view.

4.2.5 AGC Configuration

The audio pipeline includes an automatic gain controller (AGC) which is applied equally to all four processed outputs from the XVF3800.

This is controlled by four parameters: `PP_AGCGAIN`, `PP_AGCMAXGAIN`, `PP_AGCDESIREDLEVEL`, and `PP_AGCONOFF`.

- `PP_AGCGAIN` both controls and reports the current multiplicative gain applied to the output beams by the AGC. The value set as the product default is the initial value. When the AGC is active, this value is then dynamically adjusted to attempt to meet the specified output power.

- `PP_AGCDESIREDLEVEL` is the parameter that sets this desired output power. The signal power of the free-running beam is measured and compared to the value of `PP_AGCDESIREDLEVEL`, and `PP_AGCGAIN` is adjusted to attempt to meet it.

- `PP_AGCMAXGAIN` is the maximum value that `PP_AGCGAIN` may take in operation.

- `PP_AGCONOFF` determines whether the AGC is permitted to adapt or whether the value of `PP_AGCGAIN` is fixed.

Note: It is important to note that the gain specified by `PP_AGCGAIN` is always applied, regardless of the value of `PP_AGCONOFF`; `PP_AGCONOFF` will only control whether or not this value is permitted to change during operation.

To set appropriate default values for these parameters, set the device's output mux to output the free-running beam:

```
xvf_host AUDIO_MGR_OP_L 6 2
xvf_host AUDIO_MGR_OP_R 0 0
```

Initialise the parameters to sensible default values:

```
xvf_host PP_AGCGAIN 1.0
xvf_host PP_AGCMAXGAIN 1000
xvf_host PP_AGCONOFF 1
```

Play a near-end signal, such as `IEEE_269-2010_Male_mono_48_kHz.wav`, at a nominal level and at a nominal distance. The exact specification for this should be determined by the desired certification. Allow `PP_AGCGAIN` to converge on a value, record the device output, and observe the output level. Should the device output be too quiet or too loud for the desired certification specification, alter `PP_AGCDESIREDLEVEL` and allow `PP_AGCGAIN` to reconverge. Once the device output level is as desired, record the stable value of `PP_AGCGAIN`. This should then be set as the product's default value for `PP_AGCGAIN`.

To configure `PP_AGCMAXGAIN`, reduce the near-end signal by 10 dB and repeat the above process, allowing `PP_AGCGAIN` to converge on a stable value. This should become the product's default value for `PP_AGCMAXGAIN`.

4.2.6   Emphasis

The rate at which the AEC converges can be optimised by compensating for the spectral characteristics of the reference signal. If the signal has significant low-frequency energy but proportionally less high-frequency energy, this will affect the AEC's rate of convergence in the high frequencies, and therefore rate of convergence overall. An optional high shelf boost may be applied to the microphone inputs using the `AEC_AECEMPHASISONOFF` parameter.

Play a representative voice sample such as `IEEE_269-2010_Male_mono_48_kHz.wav` as the reference audio and capture the post-gain, post-delay reference signal:

```
xvf_host AUDIO_MGR_OP_L 5 0
xvf_host AUDIO_MGR_OP_R 0 0
```

Perform a Fourier transform using Audacity, or equivalent, and identify the peak magnitude value. Compare this to the magnitude of the signal at 8 kHz. It is expected that the magnitude of the signal at 8 kHz will be less than the peak magnitude. If they have similar magnitudes, set `AEC_AECEMPHASISONOFF` to 0. If the difference in magnitudes is around or greater than 8 dB, set `AEC_AECEMPHASISONOFF` to 1. If the difference is around or greater than 40 dB, set `AEC_AECEMPHASISONOFF` to 2.

The impact of tuning this parameter may be observed by measuring the AEC convergence speed. From a fresh restart, set the following parameters to output clear AEC residuals from a selected pair of microphones:

```
xvf_host PP_MIN_NS 1.0
xvf_host PP_MIN_NN 1.0
xvf_host PP_ECHOONOFF 0
xvf_host PP_NLATTENONOFF 0
xvf_host PP_AGCONOFF 0
xvf_host AUDIO_MGR_OP_L 7 [microphone number]
xvf_host AUDIO_MGR_OP_R 7 [microphone number]
```

Play a representative voice sample such as `IEEE_269-2010_Male_mono_48_kHz.wav` as the reference input on a loop for around 60 seconds. Capture the device output; these will be the AEC residuals generated. Observe the spectrogram of the output signal, and verify that the AEC converges evenly for all frequencies; that is to say, that the high frequencies converge as quickly as the low frequencies.

4.2.7 Additional Parameters

4.2.7.1 FMIN_SPEINDEX

`PP_FMIN_SPEINDEX` is a parameter that controls the frequency-dependent suppression that the device performs in a double-talk environment. In the case of double-talk, the device's output will suppress frequencies below the value of `PP_FMIN_SPEINDEX` more than frequencies above the value of `PP_FMIN_SPEINDEX`

Set the following to output clear AEC residuals from a selected pair of microphones:

```
xvf_host PP_MIN_NS 1.0
xvf_host PP_MIN_NN 1.0
xvf_host PP_ECHOONOFF 0
xvf_host PP_NLATTENONOFF 0
xvf_host PP_AGCONOFF 0
xvf_host AUDIO_MGR_OP_L 7 [microphone number]
xvf_host AUDIO_MGR_OP_R 7 [microphone number]
```

Depending on the device sample rate, copy from `/sources/app_xvf3800/tuning/` the track `white_noise_0dbfs_2ch_16khz.wav` or `white_noise_0dbfs_2ch_48khz.wav`. Play through the reference input the white noise signal, and capture the AEC residuals that are output from the device. Take a Fourier transform of the interval from 40 - 60 seconds, and plot the magnitude of the coefficients. Set `PP_FMIN_SPEINDEX` to the highest frequency after which there is no further decrease in the amplitude spectrum; in the spectrum shown in Fig. 4.7 for example, `PP_FMIN_SPEINDEX` should be set to around 1200 (1.2 kHz). If the spectrum appears roughly flat from around 500 Hz onwards, with no significant decrease in amplitude at higher frequencies, leave `PP_FMIN_SPEINDEX` at its default value of 593.75 Hz.
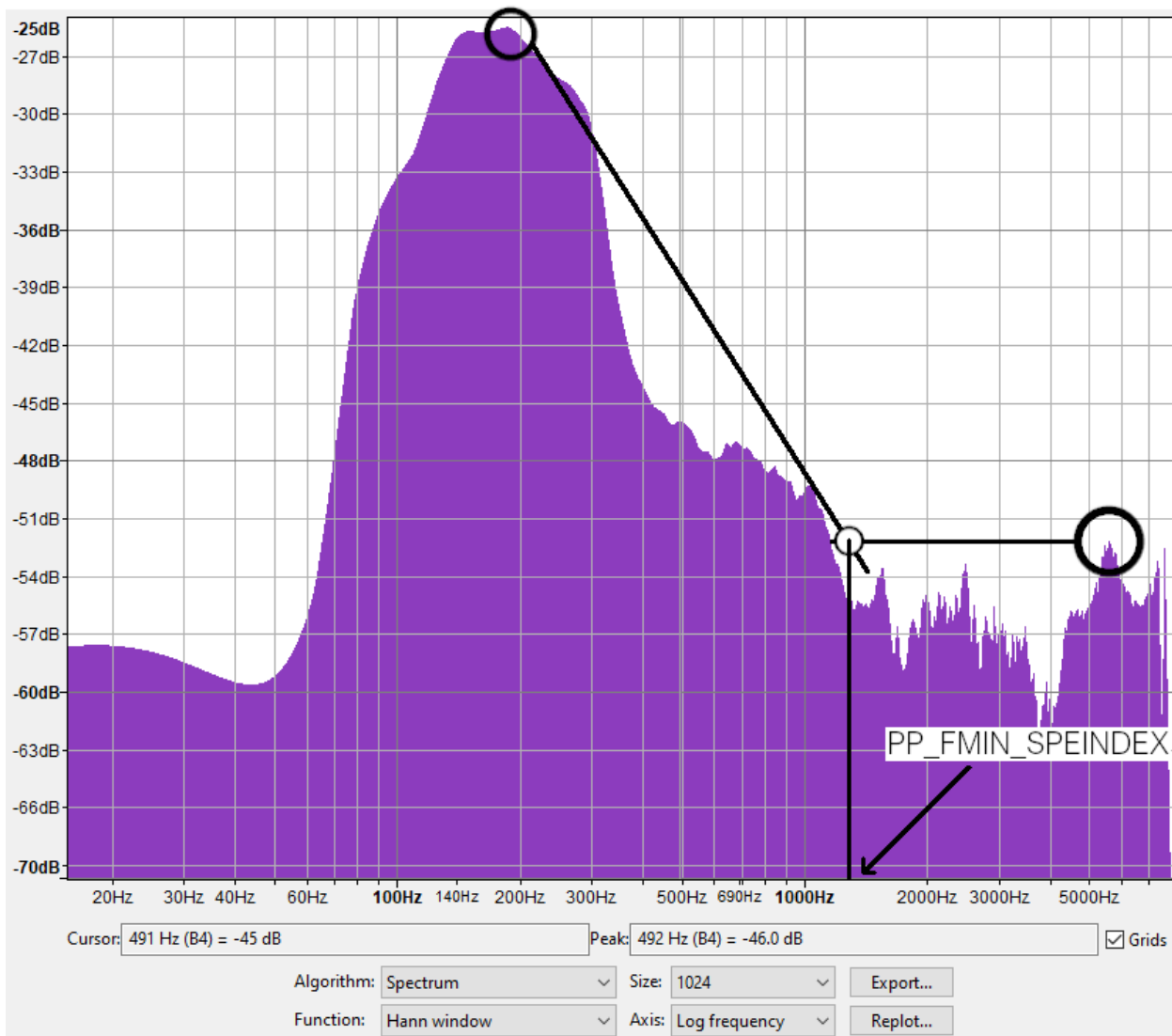
Fig. 4.7: Illustration of the method by which `PP_FMIN_SPEINDEX` may be determined

4.2.7.2 MGSCALE

The `PP_MGSCALE` parameter controls additional noise suppression that is applied during periods of far-end activity. The aim is to optimise speech clarity output from the device during periods of stationary far-end activity, while also ensuring that there is good echo suppression in periods of non-stationary far-end activity. An undesirable scenario may arise if there exists unintended low-level noise in the reference signal, from e.g. ADC noise in the reference path. In this scenario, the low-level noise may be erroneously detected as far-end speech; the device may then incorrectly detect that double-talk is present and overly suppress near-end speech. The `PP_MGSCALE` parameter configures where this trade-off between far-end echo suppression and near-end signal clarity lies.

To tune both the *min* and *max* values for the `PP_MGSCALE` parameter, set the following:

```
xvf_host PP_GAMMA_E 1.0
xvf_host PP_GAMMA_ENL 1.0
xvf_host PP_GAMMA_ETAIL 1.0
xvf_host PP_ECHOONOFF 1
xvf_host PP_NLATTENONOFF 0
xvf_host PP_MIN_NS 1.0
```

```
xvf_host PP_MGSCALE 1 1 1
xvf_host AUDIO_MGR_OP_L 6 3
xvf_host AUDIO_MGR_OP_R 0 0
```

Play a representative far-end signal such as `IEEE_269-2010_Male_mono_48_kHz.wav` on loop as the reference input. Provide a just-noticeable stationary near-end noise signal. Observe the device output, including the spectrogram. Increasing the value of *max* - the first parameter to `PP_MGSCALE` - will reduce the amount of residual echo. Increase *max* until no further improvements are observed.

Note: A typical value for *max* will be between 100 and 1000.

Set *min* to the derived value for *max* so that the two are equal.

Depending on the device sample rate, copy from `/sources/app_xvf3800/tuning/` the track `silence_2ch_16khz.wav` or `silence_2ch_48khz.wav`. Play the silence track into the reference input, and provide a representative near-end signal such as `IEEE_269-2010_Male_mono_48_kHz.wav`. Subjectively listen to the device output. There may be stationary noise present on the far-end, which may cause erroneous echo suppression and therefore erroneous speech distortion. Reducing *min* can reduce near-end speech distortion at the cost of reduced stationary noise suppression where stationary noise is present in the far-end signal.

4.2.8 Tuning the Non Linear Model

4.2.8.1 Non-linear Echo

It is likely that in all devices, regardless of the quality of the audio design, there will exist some non-linearities. The aim of non-linear estimation is to model the remaining residual echo after linear echo content (including tail echoes) has been removed. This is achieved in the XVF3800 by use of a self-training non-linear model.

It is very important to ensure that non-linear model training takes place in a silent environment, and that the environment is ideally anechoic; the RT60 of the environment for example should be as low as possible, and absolutely below 0.3s.

Note: The XVF3800 provides an estimate of the current RT60 value of its environment. This is obtained via the `AEC_RT60` command.

It is also important to minimise/eliminate any path changes in the environment during non-linear tuning, such as movement of people or objects. This tuning step is very deliberately placed after any gain or pre-processing adjustments have been made. Any changes to the device's gain structure, including changing any filtering, will require retuning of the non-linear model.

4.2.8.2 Tuning Setup for Non Linear model

This tuning process is somewhat lengthy, and so a set of files and associated training script have been provided for this tuning step. The process differs slightly depending on whether the host device can play audio directly through the device (as in Fig. 4.8) or whether a 3rd machine is required (as in Fig. 4.9).
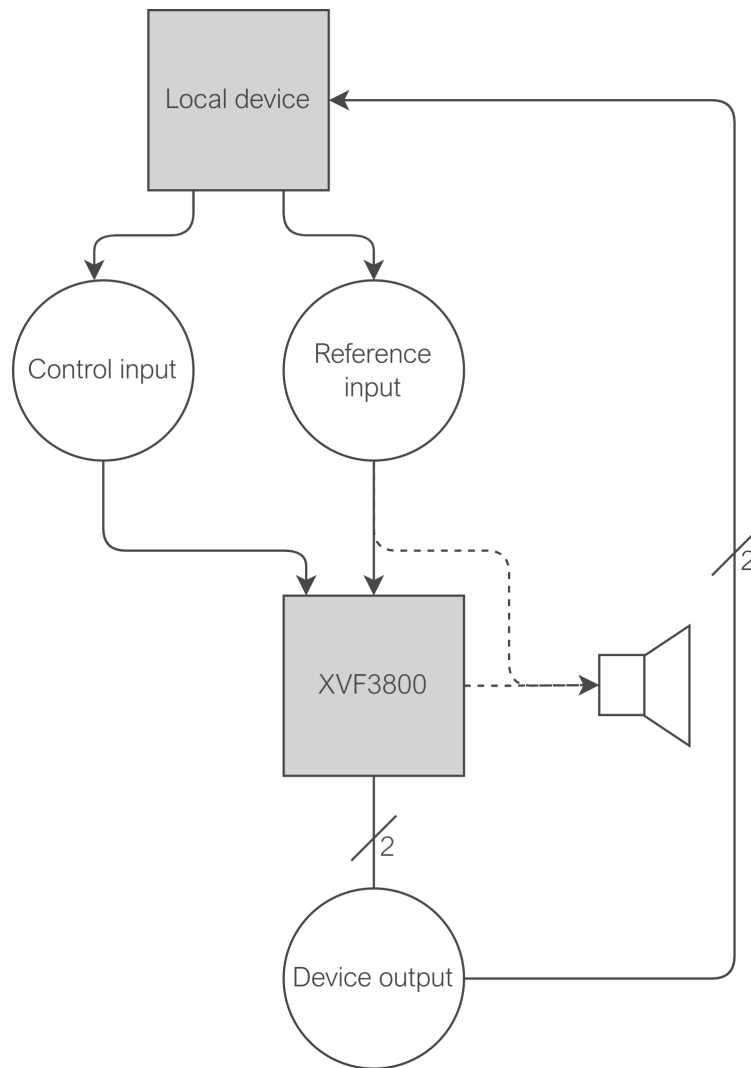
Fig. 4.8: Top-level diagram of a system where the audio host and control host are the same device

Local Device   For this route, it is assumed that the host device is also the device that is providing audio to the XVF3800, through e.g. an I^{2}S interface for the XVF3800 INT device variant or USB interface for the XVF3800 UA device. Locate the *nl_model_training.py* script provided in the `sources/app_xvf3800/nl_model_gen` directory in the release package . Run the script as:

```
python3 nl_model_training.py <host application> -p <communication protocol>
```

> **Warning:** <host application> is the `xvf_host`. If this application has not been installed so that it is visible in the operating system `$PATH`, the full path name of the `xvf_host` application must be supplied.

This will generate an output file with the default name of `nlmodel_buffer_override.bin.r16.c40`. Copy this file to `/sources/app_xvf3800/nl_model_gen/nlmodel_bin` and rerun the build process to generate a binary with this non-linear model set as default.

Refer to the docstring for this script for further guidance.
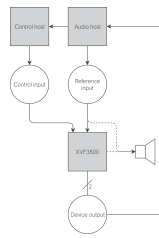
Fig. 4.9: Top-level diagram of a system where the audio host and control host are separate devices

Remote Device   For this route, it is assumed that a 3rd device is acting as the audio source (here termed the "audio host"). Therefore, to issue control commands, it is necessary to remotely connect to the "control host" (assumed to be a Raspberry Pi) over SSH. Locate the *remote_nl_model_training.py* script included in the release package. Further, locate the host application binaries on the audio host; these should be located at */host_v0.2.0/rpi* in the release package. Ensure that the audio host has passwordless access to the control host over SSH; this may be achieved by generating an SSH key pair and adding the public key to *~/.ssh/authorized_keys* on the control host.

This script requires that the audio host have an installation of `sox` in its path, as well as Python3 with `matplotlib` and `asyncssh` installed via Pip. Ensure that the default loudspeaker and microphone on the audio host are set as the device to be tuned.

From the audio host, run the script as:

```
python3 remote_nl_model_training.py <control host IP address> <host application binary␣
→path on the audio host>
```

Once the script has run, locate the generated `nlmodel_buffer_override.bin.r16.c40` and corresponding plot in the `src.autogen` directory. Copy this file to */sources/app_xvf3800/nl_model_gen/nlmodel_bin* and re-run the build process to generate a binary with this non-linear model set as default. Refer to the docstring for this script for further guidance.

4.2.8.3   Echo Suppression

With the non-linear model trained, we are now in a position to balance echo suppression against speech distortion. Five tuning parameters are relevant for this section:

- `PP_ECHOONOFF`: This parameter sets whether echo suppression is enabled or disabled overall.

- `PP_NLATTENONOFF`: This parameter sets whether non-linear echo suppression is enabled or disabled.

- `PP_GAMMA_E`: This parameter adjusts the oversubtraction factor for direct and early echo suppression.

- `PP_GAMMA_ENL`: This parameter adjusts the oversubtraction factor for non-linear echo suppression.

- `PP_GAMMA_ETAIL`: This parameter adjusts the oversubtraction factor for echo tail suppression.

For the `PP_GAMMA_*` parameters, a value of 1.0 indicates that the device has correctly estimated and suppressed the respective echo classes from the output. Increasing these values increases the amount of suppression, and indicates that the device has underestimated the amount of echo in the outputs. It is unlikely for the device to overestimate the amount of echo, and so it is not advised to set these parameters to values below 1.0. A typical range for these parameters is between 1.0 and 1.7.

Increasing these parameters will always affect the quality of the speech signal. Attempt in the first instance to create as good an acoustic design as possible, with a linear loudspeaker, good quality microphones, and a maximally non-linear enclosure. This will reduce or eliminate the need to adjust these values, and will present a more performant device.

PP_GAMMA_E and PP_GAMMA_ENL   The objective of tuning these two parameters is the removal of echoes to pass Teams EQUEST and ECC specifications. Ensure that this tuning step takes place in an anechoic or mildly reverberant environment, with an RT60 less than 0.3 s.

Set the device as follows to output the autoselect beam in the left channel and the AEC residual signal for microphone 0 in the right channel:

```
xvf_host PP_AGCONOFF 0
xvf_host PP_MIN_NN 1.0
xvf_host PP_MIN_NS 1.0
xvf_host PP_GAMMA_E 1.0
xvf_host PP_GAMMA_ENL 1.0
xvf_host PP_GAMMA_ETAIL 1.0
xvf_host AUDIO_MGR_OP_L 6 3
xvf_host AUDIO_MGR_OP_R 7 0
```

Play through the reference input a representative signal, such as `IEEE_269-2010_Male_mono_48_kHz.wav`. Allow the AEC to converge After 30 seconds, play a representative near-end signal in addition to the far-end signal, to place the device into a representative double-talk scenario. Listen to the device's output. Starting with the default value of 1.0, adjust `PP_GAMMA_E` to make the trade-off between double-talk performance and echo suppression. Should the value of `PP_GAMMA_E` need to exceed around 1.4 to achieve acceptable performance, consider adjusting `PP_GAMMA_ENL` instead, especially if the echoes that remain in the AEC residual signal are of a non-linear nature.

To identify the nature of the echoes that remain, listen to the AEC residual signal and categorise the residual echoes as follows:

- Linear residual echoes: Echoes can be understood at a low level but do not sound distorted and do not sound reverberated. Controlled by `PP_GAMMA_E`.

- Tail echoes: Echos sound reverberated, with no direct component. Controlled by `PP_GAMMA_ETAIL`.

- Non-linear echoes: Echoes sounds very distorted, with no discernable e.g. speech content. Controlled by `PP_GAMMA_ENL`.

PP_GAMMA_ETAIL   To tune this parameter, repeat the above procedure in a moderately reverberant room (with an RT60 between 0.3 and 0.9s). Clear tail echoes should be observed in the residual signal, and these tail echoes should be improved by adjustment of `PP_GAMMA_ETAIL`. Adjust this parameter, making a trade-off between double-talk performance and echo suppression.

4.2.8.4   Noise Suppression

Two parameters control suppression of stationary and non-stationary noise in the device output: `PP_MIN_NS` and `PP_MIN_NN` respectively. These parameters take values between 0 and 1, representing the multiplicative attenuation of these two noise sources. For example, `PP_MIN_NS` is set to 0.15 by default, representing a roughly 15 dB attenuation of stationary noise in the device output. It is recommended that `PP_MIN_NN` is set by default to 0.51 or higher, representing at most a 6 dB attenuation of non-stationary noise in the device output. Reducing this value further may have significant impact on near-end speech quality, especially in reverberant environments.

To tune these parameters, set the device as follows:

```
xvf_host PP_AGCONOFF 0
xvf_host PP_MIN_NS 0.15
xvf_host PP_MIN_NN 0.51
xvf_host AUDIO_MGR_OP_L 6 3
xvf_host AUDIO_MGR_OP_R 0 0
```

Play a representative near-end signal, such as `IEEE_269-2010_Male_mono_48_kHz.wav`. Subjectively evaluate the device output, first noting the presence of stationary noise. Reduce `PP_MIN_NS` to suppress this

noise further. Reducing this parameter may introduce or increase distortion in near-end speech; ensure that a balance is struck between speech quality and stationary noise suppression. Note next the presence of non-stationary noise. Reduce `PP_MIN_NN` to suppress this noise further. As with `PP_MIN_NS`, reducing this parameter may introduce distortion in near-end speech, particularly in reverberant environments. Ensure that an appropriate balance is struck between speech quality and non-stationary noise suppression.

4.2.8.5 ATTNS

The ATTNS parameters (`PP_ATTNS_MODE`, `PP_ATTNS_NOMINAL`, and `PP_ATTNS_SLOPE`) control an additional reduction in AGC gain during non-speech periods. Collectively, they attempt to combat an undesirable side-effect of the use of an AGC - the tendency to noticeably amplify noise in non-speech periods when the near-end speech signal is quiet. The Zoom Rooms specification test 7.4.3 (as of writing, last issued in October 2019) sets limits on how amplified this non-speech noise may be when the AGC is at a high gain compared to the noise level when the AGC is at a low gain. Therefore, by attenuating noise at a greater strength when the AGC is at a high gain we may reduce this noise and achieve better performance in these tests.

The overall behaviour of the ATTNS may be selected with the `PP_ATTNS_MODE` parameter, which both functions as a specifier of whether the ATTNS is in use and whether bias is applied against selecting beams with high noise as the autoselected beam.

- 0 - The ATTNS is off

- 1 - The ATTNS is on, with an additional check to ensure no beam with high noise is selected as the autoselect beam (recommended if in use)

- 2 - The ATTNS is on, with no additional check

When the ATTNS is on, `PP_ATTNS_NOMINAL` and `PP_ATTNS_SLOPE` control the additional attenuation proportional to:

$$\text{ATTNS\_NOMINAL} * \left(\frac{\text{AGCGAIN\_INIT}}{\text{AGCGAIN\_CURRENT}} \right)^{\text{ATTNS\_SLOPE}}$$

where AGCGAIN_INIT is the value of `PP_AGCGAIN` set as the default value at initialisation, and AGCGAIN_CURRENT is the current value of `PP_AGCGAIN`. Because of this module's relationship with the current value of `PP_AGCGAIN`, this module has no effect when `PP_AGCONOFF` is set to 0.

Because both the Teams v4 and Zoom Rooms specifications specify this suppression as a ratio between the noise at a nominal speech level and the noise at a low speech level, it may be required to tune both of these parameters in parallel; changing one may have an effect on the required value for the other, and vice-versa.

To tune these parameters, ensure that `PP_AGCGAIN` and `PP_AGCMAXGAIN` are tuned correctly, then perform the following:

ATTNS_NOMINAL Issue the following to set appropriate default values for this tuning step:

```
xvf_host PP_AGCONOFF 1
xvf_host PP_ATTNS_MODE 1
xvf_host PP_ATTNS_NOMINAL 1.0
xvf_host PP_ATTNS_SLOPE 0.0
```

Play a representative near-end signal at a nominal level, such as the ITU P.501 7.3.2 reference signal `FB_male_female_single-talk_seq.wav`. This signal can be found in the `Speech Signals/ Test_Signals_Clause 7/Speech Test Signals Clause 7.3 & 7.4/English_FB_clause_7.3/` directory of the ITU P.501 download package.

Setting ATTNS_NOMINAL > 1 should provide more noise suppression during silence. With a particular specification in mind, increase this value until desired/specified noise suppression is achieved during the test conditions. For example, this could be done by monitoring average A-weighted noise during the period 1s after the end of a sentence in the reference signal and ensuring that it is within satisfactory bounds; this is

usually specified as a ratio between this value and the averaged value obtained with the near-end signal at a range of low levels.

ATTNS_SLOPE   Issue the following to set appropriate default values for this tuning step:

```
xvf_host PP_AGCONOFF 1
xvf_host PP_ATTNS_MODE 1
xvf_host PP_ATTNS_NOMINAL <default found in previous step>
xvf_host PP_ATTNS_SLOPE 1.0
```

Play a representative near-end signal at a nominal level, such as the ITU P.501 7.3.2 reference signal `FB_male_female_single-talk_seq.wav`. Setting ATTNS_SLOPE > 1.0 provides additional noise suppression during silence, proportional to an increased AGC gain. With a particular specification in mind, increase this value until desired/specified noise suppression is achieved during the test conditions. For example, this could be done by monitoring average A-weighted noise during the period 1s after the end of a sentence in the reference signal and ensuring that it is within satisfactory bounds; this is usually specified as a ratio between this value and the averaged value obtained with the near-end signal at a range of low levels.

4.2.8.6   Path Change Detection

The XVF3800 provides a facility to detect significant path changes in the device's environment such as handling the device and moving to a different location using a module called the Path Change Detector (PCD). If a path change is detected, heavy near-end suppression during far-end activity is applied in order to allow the AEC time to reconverge to its new environment. If the device incorporating the XVF3800 is not intended for a mobile application (for example, a wall-mounted sound bar), then detection of path changes is not necessary.

The PCD may be tuned using the `AEC_PCD_COUPLINGI`, `AEC_PCD_MINTHR`, and `AEC_PCD_MAXTHR` parameters.

`AEC_PCD_COUPLINGI` controls the rate of detection of a path change, and takes a value between 0 and 1. Setting this to a low value encourages fast detection of path changes at the increasing risk of false positives during double-talk. Setting this to a high value slows detection of path changes (and increases the detection threshold, meaning some small changes may be missed) but reduces the risk of false positives in double-talk. Setting this parameter to a value outside of the range 0 to 1 will disable the PCD. Tuning of this parameter is necessarily very situation- and product-dependent. Monitoring of the `AEC_AECPATHCHANGE` parameter can allow insight into whether a path change has been detected; reading a 1 value implies that a path change has recently been detected and that the device output is currently heavily suppressed during far-end activity. This parameter will reset to 0 after the AEC has reconverged.

`AEC_PCD_MINTHR` and `AEC_PCDP_MAXTHR` are used to set sensitivity thresholds, and their use depends on the overall Echo Return Loss Estimate (ERLE) of the device. For devices with a high ERLE (implying a high ratio between the provided reference signal and the resultant AEC residual, and therefore high cancellation), use `AEC_PCD_MINTHR` to limit the lower bound. Decreasing this value from its default of 0.02 will increase the sensitivity of the PCD. For devices with a low ERLE, use `AEC_PCD_MAXTHR` to limit the upper bound. Decrease this value from its default of 0.2 to increase the sensitivity of the PCD.

4.3   Changing Default Parameter Values

The default parameters set at start-up are loaded from the file `product_defaults.c` in `sources/app_xvf3800/src/default_params`. In this file the values are included from some header files auto-generated at compile time. The values used in `product_defaults.c` must be updated using the YAML files stored in `sources/app_xvf3800/cmd_map_gen/yaml_files/defaults/`.

Note: Any default value set outside the YAML files will be overwritten at compile time.

In the `defaults` folder four files used to generate `product_defaults.c` are present:

- `mic_geometries.yaml`

- `control_param_values.yaml`

- `gpi_config.yaml`

- `gpo_config.yaml`

> **Warning:** All the parameters in the files above must be set; failure to do this can lead to unexpected behaviour of the device, such as uninitialized start-up values.

`mic_geometries.yaml` contains the coordinates for each of the 4 mics for both the linear and squarecular geometries. An example of the values is below:

```
LINEAR_GEOMETRY:
  - MIC0: ( -0.04995f,    0.00f,      0.00f )
  - MIC1: ( -0.01665f,    0.00f,      0.00f )
  - MIC2: ( 0.01665f,     0.00f,      0.00f )
  - MIC3: ( 0.04995f,     0.00f,      0.00f )
SQUARECULAR_GEOMETRY:
  - MIC0: ( 0.0333f,      -0.0333f,   0.00f )
  - MIC1: ( 0.0333f,      0.0333f,    0.00f )
  - MIC2: ( -0.0333f,     0.0333f,    0.00f )
  - MIC3: ( -0.0333f,     -0.0333f,   0.00f )
```

The user must update the values to match the geometry used in their target application. The default configurations shown above are those supported by the XK-VOICE-SQ66 development kit.

`control_param_values.yaml` lists all the control parameters which can be configured. An example of a parameter with a default value is below:

```
PP_RESID:
  - cmd: PP_AGCONOFF
    default_value: on
```

The parameters in the file are organized into arrays, and each array contains all the parameters related to a particular control resource ID. The parameter name is stored in the `cmd` key and the default value in the `default_value` key. In the example above, the default value of the parameter *PP_AGCONOFF* belonging to the *PP_RESID* is *on*. The number of values and type of each parameter may vary from command to command. It is advised to look up the command information in the tables in the Appendix of User Guide and to follow the format of the original default values in order to set the values properly.

`gpi_config.yaml` stores all the settings of the GPI pins. The XVF3800 has 2 configurable GPI pins and the following parameters can be modified:

- *active_level*: 0 for low and 1 for high

- *event_config*: four types of events are supported:

  - *EdgeNone*: no event is detected on either edge

  - *EdgeFalling*: an event is detected on the falling edge (high to low transition)

  - *EdgeRising*: an event is detected on the rising edge (low to high transition)

  - *EdgeBoth*: one event is detected on the rising edge and one on the falling edge

The default configurations of the GPI pins are below:

```
# Exactly GPIO_NUM_INPUT_PINS pins should be defined here

PIN0:
  active_level: 1
  event_config: EdgeNone
PIN1:
  active_level: 1
  event_config: EdgeNone
```

`gpo_config.yaml` lists the settings of all the GPO pins and ports. The XVF3800 device has one 8-bit port designated for GPO. Only five of the eight are pinned out, and three pins are required for the device to operate normally, leaving the remaining two pins available for user modification. These pins are number 6 and 7, and they are used to control the LEDs in the default firmware.

In the file each port must be listed; the XVF3800 only implements *PORT0*. For each port an array of eight pins must be defined and each pin has the following configurable settings:

- *pin_number*: this value shouldn't be modified

- *active_level*: 0 for low and 1 for high

- *output_duty_percent*: Pulse-width modulation (PWM) duty cycle specified as a percentage

- *flash_serial_mask*: serial flash mask where each bit specifies the GPO pin state for a 100 ms time period interval

The default configurations of the GPO port and pins are below:

```
PORT0:
  # UNUSED: NOT PINNED OUT
  - pin_number: 0
    active_level: 1
    output_duty_percent: 0
    flash_serial_mask: 0xFFFFFFFF
  # UNUSED: NOT PINNED OUT
  - pin_number: 1
    active_level: 1
    output_duty_percent: 0
    flash_serial_mask: 0xFFFFFFFF
  # UNUSED: NOT PINNED OUT
  - pin_number: 2
    active_level: 1
    output_duty_percent: 0
    flash_serial_mask: 0xFFFFFFFF
  # GPO_DAC_RST_N_PIN
  - pin_number: 3
    active_level: 1
    output_duty_percent: 100
    flash_serial_mask: 0xFFFFFFFF
  # GPO_SQ_nLIN_PIN
  - pin_number: 4
    active_level: 1
    output_duty_percent: 0
    flash_serial_mask: 0xFFFFFFFF
  # GPO_INT_N_PIN
  - pin_number: 5
    active_level: 1
    output_duty_percent: 100
```

```
    flash_serial_mask: 0xFFFFFFFF
  # GPO_LED_RED_PIN
  - pin_number: 6
    active_level: 0
    output_duty_percent: 0
    flash_serial_mask: 0xFFFFFFFF
  # GPO_LED_GREEN_PIN
  - pin_number: 7
    active_level: 0
    output_duty_percent: 0
    flash_serial_mask: 0xFFFFFFFF
```

Note: When the default parameters are changed, it is necessary to rebuild the application and reload onto the XVF3800 as described in the following section. See *Building an Executable*.

5 Building the Application Firmware

5.1 Introduction

The XVF3800 comprises a specialised xcore.ai processor and a firmware executable. A set of firmware images is provided in the binary release package which are configured to run correctly on the XK-VOICE-SQ66 development kit. However, when using the XVF3800 in a product design it is normally necessary to modify the firmware to match the hardware and to configure a number of settings. This is achieved by modification of the configuration files supplied as source code and rebuilding the modified code to create a new firmware image.

Instructions on configuring the firmware is included in the *Tuning the application* section.

This section explains how to build the XVF3800 application from the source files.

5.2 Prerequisites

The XVF3800 source code can be build on Windows, MacOS and Linux platforms.

Note: An active internet connection is required as part of the process as the build scripts download additional packages to configure the environment.

The XVF3800 build procedure requires that a set of 3rd Party software packages, listed below, are installed on a development computer before attempting to build firmware for the XVF3800.

5.2.1 Python3

A standard installation of Python version 3.9 or higher should be present on the development machine. This is available by default on some platforms, but if required it can be installed from https://www.python.org/downloads/.

The pip3 package manager included in the standard python configuration is used to install some tools and python is required to run some setup and tuning tools.

5.2.2 XMOS tools

XTC Tools 15.2.x : This is the XMOS toolchain which allows users to compile, link, deploy and debug applications on all XMOS processors.

The XTC Tools can be downloaded from https://www.xmos.ai/software-tools/ and installed on a development machine following the instructions in the https://www.xmos.ai/view/Tools-15-Documentation. XTC tools can run on Windows, MacOS and Linux platforms.

5.2.3 Build Tools

CMake >= version 3.21.0 : CMake is a build tool for managing application compilation.

The CMake tool package can be installed following the instructions for your specific OS platform at https://cmake.org/install/.

The recommended make tools depend on the development platform:

(Linux and MacOS): GNU Make - Available through your system package manager.

Ninja (Windows): Ninja - Follow install instructions at https://ninja-build.org/ or on windows install with `winget` by running the following commands in PowerShell:

```
# Install
winget install Ninja-build.ninja
# Reload user Path
$env:Path=[System.Environment]::GetEnvironmentVariable("Path","User")
```

5.3   XVF3800 Release Package

The XVF3800 firmware is supplied in two different release packages which are distributed as ZIP archive files with the following contents:

1. Binary release - a set of pre-compiled images that will run on the XK-VOICE-SQ66 development kit

2. Source release - Source code and libraries to allow customisation of the XVF3800

This section describes how to use the second package.

Release packages can be obtained from https://www.xmos.ai/ or your XMOS representative.

Load ZIP archive onto your development platform and expand the archive into a convenient directory. The contents of the source release package are shown below:

```
.
├── CHANGELOG.rst      <- list of changes of current and past releases
├── LICENSE.rst        <- license file
├── precompiled        <- folder containing the precompiled libraries
├── README.md          <- readme file
└── sources            <- source files necessary to build  XVF3800 applications
```

The user modifiable code is found in the `sources/app_xvf3800` folder.

5.3.1   Standard Configurations

The XVF3800 release package contains a set of standard build configurations that will suit the majority of use cases for the XVF3800 device. The table below lists the key configuration parameters.

Table 5.1: Build-time combinable parameters

Parameter	Options	Abbreviation	Notes
Device configuration	INT-Device USB	-intdev -ua	
I^2S LR clock rate	16000 48000	-lr16 -lr48	Select sampling rate for I^2S interface; ignored for UA configurations.
USB IN/OUT sample rate	16000 48000	-io16 -io48	Only valid in UA configuration; ignored for INT configurations.
Microphone geometry	Linear Square or circular	-lin -sqr	Selects microphone configuration on XK-VOICE-SQ66 board
Control protocol	I^2C SPI	-i2c -spi	Select control protocol for INT configurations; ignored for UA configurations, as they always use USB as control interface.
Audio MCLK	Use external MCLK signal	-extmclk	Only valid on INT-Device configuration. Should be omitted if an external MCLK is not used.

5.3.2 Image Names

The XVF3800 built image names comprise the parameter abbreviations listed above in a set order:

```
|project|-<device config>-<sample rate>-<mic geometry>-<control
protocol>-<audio MCLK>
```

Examples:

```
xvf3800-ua-io48-lin
```

```
xvf3800-ua-io16-sqr
```

```
xvf3800-intdev-lr48-lin-spi-extmclk
```

5.4   Build Process

5.4.1 Set up the environment

Note: The software packages in the *Prerequisites* section must be installed before starting this process.

To build the XVF3800 application, open a command-line terminal and ensure that the XMOS tools are config-
ured in the environment. This can be checked by typing:

```
xcc --version
```

which should display the tools version information. If this does not happen please consult the `Configuring
the command-line environment` section of https://www.xmos.ai/software-tools/ at https://www.xmos.ai/
view/Tools-15-Documentation.

5.4.2 Configure the build system

The build process comprises of two phases - the first phase sets up the build environment and downloads
key components while the second phase builds a specific executable. Configuration is only required to be
done once.

To set up the environment change directory to the `sources` directory of the release package and install the
required Python3 packages:

```
cd sources
```

```
pip3 install -r requirements_build.txt
```

Then configure the build. This step can take several minutes. The preset to use depends on the host machine.

```
# Windows
cmake --preset=rel_app_xvf3800_windows
# Not Windows
cmake --preset=rel_app_xvf3800
```

When this is complete the XVF3800 build system will have been configured.

5.4.3 Build an executable

The next phase is to build a specific executable. The release package contains a set of preset configurations
that cover the main use case for the XVF3800 device.

To see the available build presets use:

```
cmake --build --list-presets
```

The naming scheme for these presets is defined in the *Standard Configurations* section above.

Select the preset you wish and start the build. For example to build a 16 kHz I^{2}S configuration with a square
microphone array and I^{2}C control the command is:

```
cmake --build --preset=intdev-lr16-sqr-i2c
```

The script will compile all the source file and when it completes the generated binary file is saved in the
subdirectory `output`.

The required executable binary will be named `application_xvf3800_intdev-<build options>.xe`. As an
example the result of the build command above would be:

```
output/application_xvf3800_intdev-lr16-sqr-i2c.xe.
```

5.5 Installing the Executable Image

Two methods exist to install on the XVF3800 hardware the executable image created in the previous section, using the `xrun` and `xflash` tools that are suppled in the XTC Tools suite.

Both methods require a connection between a development computer and the XVF3800 via an XTAG4 debug adapter. Instructions to set up the XK-VOICE-SQ66 development kit can be found in the *Setting up the hardware* section of this guide.

5.5.1 Install Using xrun

The `xrun` tool loads the executable image into the XVF3800 RAM without storing it in the XVF3800 Flash ROM. It then starts the operation of the XVF3800 using this executable image. Using the example from above the required command is:

```
xrun application_xvf3800_intdev-lr16-sqr-i2c.xe
```

5.5.2 Install Using xflash

The `xflash` tool stores the executable image in the XVF3800 Flash ROM. A subsequent power-cycle loads the stored image into the XVF3800 RAM and starts its operation. Using the same example from above the required command is:

```
xflash application_xvf3800_intdev-lr16-sqr-i2c.xe
```

5.6 Using SPI Boot

5.6.1 Creating a SPI Boot File

To use the built `application_xvf3800_[..].xe` executable generated from the above process as a SPI boot image it is necessary to convert the `.xe` image into a `.bin` file that includes a bootloader.

The files required to build this `.bin` image can be found in the `sources/modules/fwk_xvf/modules/boot_spi_slave` directory of the release package.

> **Warning:** This procedure requires the XTC tools to be installed on a development computer. It cannot be run on the Raspberry Pi host.

To create a SPI boot image change into the scripts folder:

```
cd sources/scripts
```

and then generate the binary image using the following command:

```
python3 generate_image.py path/to/application_xvf3800_[..].xe
```

An SPI bootable file `application_xvf3800_[..]_spi_boot.bin` will be created in the `output` subdirectory.

5.6.2 Using a SPI Boot File

The SPI boot process is documented in the XVF3800 datasheet. An example python script is provided in the release package which uses a Raspberry Pi to transfer the image and boot the XVF3800 device on an XK-VOICE-SQ66 development kit.

Note: The Raspberry Pi must be setup as described in the `setting up the hardware` section for this script to operate correctly.

The python script can be found in `sources/modules/fwk_xvf/modules/boot_spi_slave/scripts` subdirectory. This script and the `_spi_boot.bin` image generated using the procedure described above should be copied onto the Raspberry Pi.

To boot the XVF3800 device the following command should be run.

```
sudo python3 send_image_from_rpi.py output/application_xvf3800_[..]_spi_boot.bin
```

The example script will need to be modified, if this procedure is executed on other host systems.

6 Some Acoustic Design Guidelines

This chapter presents a brief guide to a number of introductory acoustic considerations that designers should take into account when integrating the XVF3800 into their end product.

It should be stressed that a more ideal acoustic design will result in fewer compromises needing to be made whilst configuring the XVF3800. Designers should invest time in the acoustic design of the end product in order to optimise the overall product performance.

Note that the requirements discussed here are only intended as general guidelines to help devise a more precise product specification. The actual specification will depend on the intended application. For example, even within a telecommunications application, a handset, personal speakerphone, or shared space speakerphone will each have very different technical specifications.

Additionally, many certification requirements have multiple available levels of certification. For example, Microsoft Teams Audio Requirements has a basic and a premium certification level, where the premium level features more stringent requirements. A similar situation with basic and premium criteria can be found in Amazon's certification requirements for voice services.

Therefore, this document will not cover all the different potential requirements or applications. These must be decided on a case-by-case basis to match with the intended application of the product with consideration to the desired certification requirements.

Instead, this document will cover the technical areas that should be considered when designing the product specification. Where numerical values are given, they should not be considered recommended or required, but instead are provided as a "ball-park" figure for context. These figures have been chosen based on a generic product design for a typical smart speaker or personal speakerphone which aims to pass the basic level of certification requirements.

6.1 Microphones

The XVF3800 requires 4 microphone inputs. These microphones may be omnidirectional; no additional benefit has been observed from the use of e.g. cardioid polar patterns.

Microphones chosen for a design should exhibit a signal-to-noise ratio (SNR) greater than 67 dB. This ensures a sufficiently low microphone self-noise, allowing a low enough noise floor for the XVF3800 to function effectively. Matched microphones are however not necessary. Total Harmonic Distortion (THD) should be less than 1%, although with modern MEMS microphones this is usually the case so long as the microphone is not operating near its acoustic overload point.

For compatibility with the XVF3800, microphones chosen should be digital MEMS microphones with a PDM output. These will be clocked at 3.072 MHz, with a decimation factor applied in firmware to generate the sampling rate used internally.

With loudspeakers operating at their loudest volume, microphones should not reach acoustic overload. At loudest loudspeaker volume, a headroom of 6 to 10 dB is a reasonable goal. It is important that the microphones are not driven into a non-linear response due to the volume of the loudspeakers in the end product. For far field voice a sensitivity of approximately -30 dBFS @ 94 dBSPL would be appropriate. For the SNR of 64 dB listed above, this would mean the noise floor is lower than -94 dBFS.

The XVF3800 supports both circular and linear microphone arrays. It is important to differentiate between "spacing" of the microphones and "aperture" of the array. The spacing is the distance between individual microphones whereas aperture is the outer limits of the array. The spacing is important for the high frequency limit, you need to make sure d < wavelength / 2 to avoid spatial aliasing, so for a 3 cm spacing the high frequency limit is 5.7 kHz.

The aperture defines the low frequency limit, based on the ability to measure a phase difference between the microphones. So when the wavelength is much larger than the array aperture, the phase (and signal) at each microphone is nearly identical and its impossible to differentiate between the two signals.

There is not a hard limit in the way there is for high frequency; usually the low frequency limit is defined based on the beam width, which also varies with the number of microphones. As a rough rule of thumb, the array aperture should be at least 5% of the wavelength, i.e., for a 10 cm aperture the low frequency limit is 172 Hz.

However, regardless of the geometry chosen, at least 2 (and preferably more) of the microphones in the array should be approximately 10 cm apart. This is in order to ensure sufficient phase difference between microphones at lower frequencies to allow the AEC to function.

The frequency response should cover the desired voice band. For a wideband application, a response from 100 Hz to 10 kHz should be easily obtainable. A frequency mask can be defined to ensure flatness of the response, e.g., ±2 dB from 100 Hz to 6 kHz and ±4 dB from 6 kHz to 10 kHz.

With zero input (i.e. a silent room), there should be low coherence between microphone signals - that is to say, the self-noise of the microphones chosen should not be correlated between microphones. If correlation is observed with zero input, this usually indicates that there exists some common-mode interference between the microphone signals. The presence of correlated noise has a negative effect on the performance of the XVF3800, and so this should be as minimal as possible. To estimate coherence between pairs of microphones at frequencies up to the Nyquist limit (which in this system will be 8 kHz), the provided *coherence.py* script can be used to generate a plot similar to that shown in Fig. 6.1, where the blue line shown is real data from two microphones in a silent room and the red line is a theoretical coherence plot between two perfect microphones measuring diffuse noise. This theoretical model is a sinc^2 function with its maximum at DC and its first zero crossing at *f* given by *f = c / 2d*, where *c* is the speed of sound (in m/s) and *d* is the distance between microphones (in m). The *coherence.py* script may be used as:

```
python3 coherence.py <mic0_1.wav>
```

The signal *mic0_1.wav* should be a 2 channel, 16 kHz WAV file with two microphone signals, which should be captured in silence; to capture these signals using the XVF3800's output, issue:

```
xvf_host AUDIO_MGR_OP_L 1 0
xvf_host AUDIO_MGR_OP_R 1 1
```

Record 30 seconds of output from the device, and repeat for the other microphones:

```
xvf_host AUDIO_MGR_OP_L 1 2
xvf_host AUDIO_MGR_OP_R 1 3
```

Further information on the use of the host application to capture output can be found in *Using the Host Application* and documentation of this script may be found in its docstring.

For optimal algorithmic performance, the coherence between each possible pair of microphones should be less than 0.1. All possible pairs of microphones should be tested; this will result in a total of 6 plots.
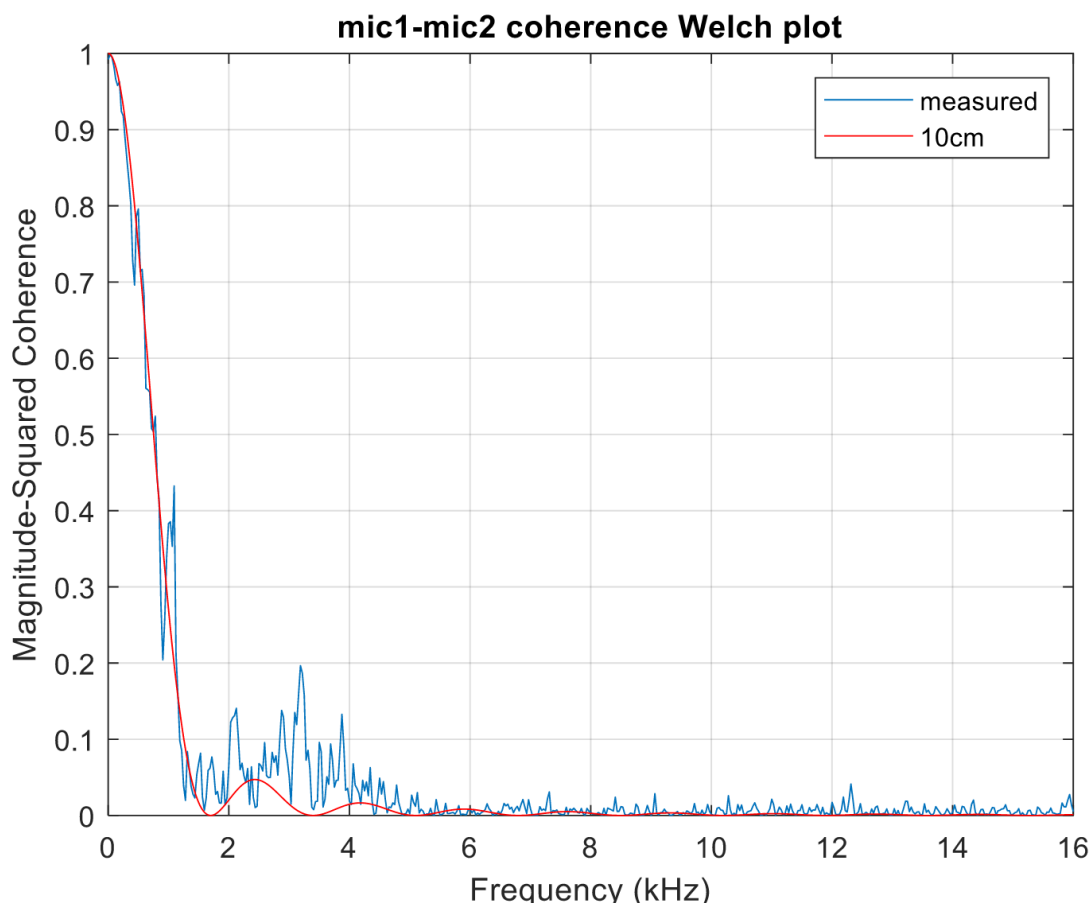
Fig. 6.1: Sample coherence plot between two microphones, where the blue line is real data and the red line is a theoretical coherence between two perfect microphones recording diffuse noise

6.2   Loudspeaker(s)

The loudspeaker, power amplifier and DAC are considered together as the playback path of the product. The frequency response of this assembly should cover the desired voice band, e.g., wideband should cover 250 Hz to 6.3 kHz. As mentioned in the microphone section, a mask can be defined to ensure desired flatness of the response within the passband. For a product which can also play music etc., the loudspeaker can have a wider frequency response; this should not degrade the performance of the voice pipeline.

The sensitivity should be sufficient for the application; it should be capable of producing 75 dB SPL at the user's ear location when a -18 dBFS signal is supplied. This can be considered the nominal sensitivity of the product. For a more general application, the maximum volume produced can be higher. However, much higher volumes will impact the ability to detect voice during playback.

The directivity of the loudspeaker should not affect the voice pipeline performance significantly. However, it may be possible to use a directional loudspeaker to achieve required levels at the user position while minimizing the feedback path to the microphones, and hence improving echo performance.

The most pressing consideration when incorporating loudspeakers into a design using the XVF3800 is the minimisation of non-linearities within the design. Whilst the XVF3800 features a linear echo canceller (the AEC), and whilst it can also suppress tail echo and non-linear echo, it is advisable to keep any non-linearities in the design to a minimum in order to guarantee optimal intelligibility and algorithmic performance.

The two main sources of non-linearity in a design arise from mechanical coupling between a loudspeaker and the microphones and from non-linearities present in the loudspeaker/amplifier stage itself. Efforts should be made to ensure that any loudspeakers are appropriately isolated from the microphones and placed physically

as far away as feasible. Isolation may take the form of mechanical decoupling from the rest of the enclosure and/or the use of soundproofing material between loudspeakers and the microphones. Additionally, product enclosures should be designed in such a manner as not to introduce non-linear effects; they should not rattle, click, vibrate, or otherwise introduce extraneous noise during normal operation.

Non-linearities present in the loudspeaker/amplifier stage are more difficult to provide generalised advice on.

Loudspeakers and amplifiers should be specified such that at nominal operating volume they are both operating within their linear region; this usually pushes design decisions towards larger or more powerful loudspeakers. As noted in the previous section, the loudspeakers at their maximum level should not be so loud that they push the microphones in the design to acoustic overload.

A THD of below 3 to 5%, measured over the full frequency range at the maximum level, is desirable. Designers should note that the THD for loudspeakers is typically only specified in datasheets at 1 kHz. THD can also be introduced by the amplifier used; it is important that amplifiers are chosen such that the overall THD of the loudspeaker system is minimised wherever possible.

Finally, it is important to consider the effect of loudspeaker placement on the far-field sensitivity of the device's microphones. In general for a given nominal level, the closer a microphone is placed to a loudspeaker the lower its gain must be in order to avoid clipping. This means that the closer a loudspeaker is located to a microphone, the lower the overall system gain will be, and therefore the lower the far-field sensitivity of the device.

6.3   Enclosure and mounting

The mechanical design of the product should account for the following:

The mounting of the loudspeaker and microphones should allow sound to propagate to and from the user without significant impediments, such that the performance of the transducers is not degraded.

The feedback path from the loudspeaker to the microphones should be minimized. The actual value of the attenuation required will vary depending on the application, but generally should be as high as possible to reduce echo. Several possible paths are present and must be considered:

- The acoustic path outside of the enclosure can be minimized by increasing the distance between loudspeakers and microphones or by choosing a more highly directional loudspeaker,

- The acoustic path inside the enclosure can be minimized by sealing the rear chambers behind the loudspeakers and microphones, and

- The vibration path in the physical structure of the enclosure can be minimized by mechanical design of the housing structure, choice of materials, and mounting of the transducers.

Vibration of loose parts in the product can cause rattles and buzzes which creates non-linearity and will drastically reduce echo performance. Ensure all components, panels and connectors are fixed.

Microphones should be prevented from detecting any additional noise sources from the product, e.g., cooling fans.

The specifics of enclosure design, in particular relating to the acoustic performance of loudspeakers, is beyond the scope of this document. There is a wealth of literature covering this topic as well as consultancy services for design.

More detailed information on transducer mounting and handling can be found from the component manufacturers. They will also provide guidance on handling during manufacture and other important considerations.

Additionally, specification requirements for various product applications can be found in certification documents. These certifications will allow the product to bear a mark which demonstrates that the product is suitable for use with a particular service.

7 DFU operations (UA only)

The Device Firmware Update (DFU) allows updating the firmware of the device from a host machine. The XVF3800 v2.0.0 supports only upgrades for the UA device over USB Endpoint 0.

Only one upgrade image may be transferred to the flash of the XVF3800. The first upgrade image will be replaced by a subsequent upgrade process. The factory image cannot be overwritten and in case of DFU failure, the factory settings will be restored after rebooting the device.

7.1 Setting up the host machine

The XVF3800 firmware is compliant with *dfu-util*. This is a host side implementation of the DFU 1.0 and DFU 1.1 specifications of the USB forum. The software is licensed under the GPL version 2. The instructions to download and install the application can be found in https://dfu-util.sourceforge.net/. The XVF3800 is compatible with the default release binaries from https://dfu-util.sourceforge.net/releases/. The host app is available for Linux, MacOS, Raspberry Pi OS and Windows.

After installing the `dfu-util(.exe)` application, only Windows requires some extra steps to install the correct USB driver. The following instructions must be followed:

1. Connect the XVF3800 board to the host PC using a USB cable.

2. If some USB driver has been installed for the *XMOS Control (Interface 3)* as described in *xvf_host application setup*, the device must be disabled by using *Device Manager*:

   a. Open *Device Manager* and find the *XMOS Control (Interface 3)*, this is usually located under the *libusb-win32 devices*.

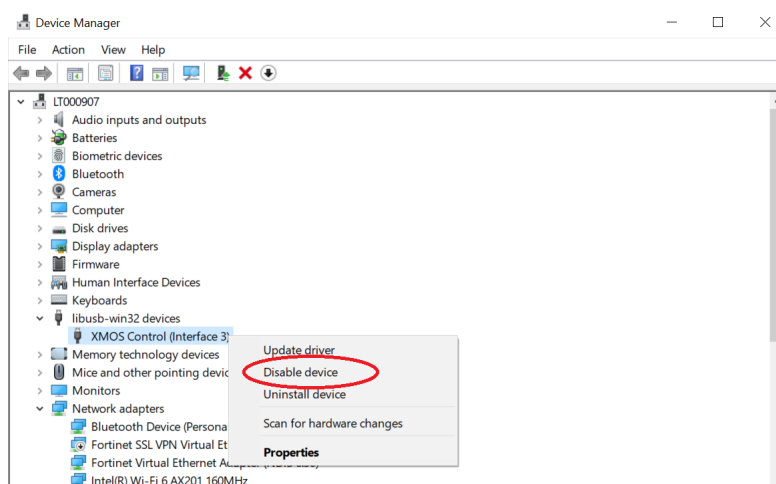   b. Right click on the interface and select *Disable device* as shown below:

Fig. 7.1: Disable the control device in Device Control

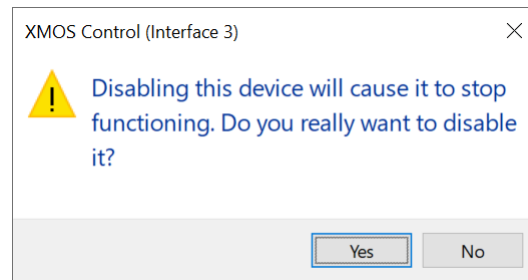   c. Click *Yes* if the message below appears:

Fig. 7.2: Accept warning message

Note: The driver can be re-enabled again by selecting *Enable device* after a right-click on the device in *Device Manager*

3. Download the third-party installation tool *Zadig* from https://zadig.akeo.ie/.

4. Open *Zadig* and select *XMOS DFU Factory (Interface 4)* from the list of devices. If the device is not present, ensure *Options -> List All Devices* is checked.

5. Select *WinUSB* from the list of drivers.

6. Click the *Install Driver* button and wait for the installation to complete.
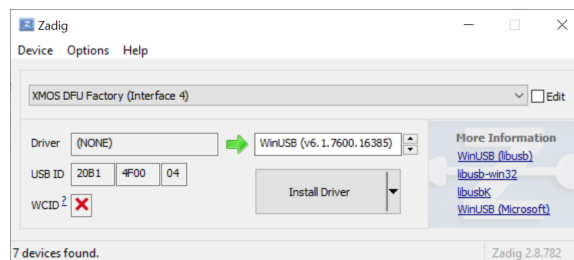
Fig. 7.3: Selecting the *WinUSB* driver in Zadig for the DFU Interface

7.2 Generation of Binary Upgrade Image

The upgrade executable (.xe) needs to be converted into a binary format. Use *xflash* and the following command to convert the .xe application into a binary form:

```
xflash --noinq --factory-version 15.2 --upgrade [UPGRADE_VERSION] [UPGRADE_EXECUTABLE] -
→o [OUTPUT_BINARY]
```

Specify `--factory-version` value of 15.2 for all 15.2.x releases of the XTC tools. (The 15.2 value refers to boot loader API for the XTC tool chain).

Note: Should a different version of the XTC tools be used in a future firmware release, the tools version number should be noted such that an update image of compatible format can be created. The `--factory-version` must match the tools version used to build the factory image.

The `UPGRADE_VERSION` number is the 16-bit format``0xJJMP`` of the executable firmware where

- J is major

- M is minor

- P is patch

`UPGRADE_EXECUTABLE` is the path to the .xe application to convert. `OUTPUT_BINARY` is the path to the binary to generate.

The *dfu-util* application issues a warning if a suffix is not added to the upgrade binary. The suffix can be added using the *dfu-suffix* host application from https://dfu-util.sourceforge.net/dfu-suffix.1.html.

7.3 Examples of DFU operations

Using the *dfu-util* application the following operations can be performed.

Note: All the commands in this section use the default USB Vendor and Product ID of the XVF3800-UA device: 0x20b1 and 0x4f00. The customer will need to use the correct ones if they are changed.

The options used in the rest of the section are described in more detail in the help menu of `dfu-util(.exe)`:

```
dfu-util(.exe) -h
```

7.3.1 Download an upgrade image to the device

The DFU download operation allows the user to upgrade the firmware of the XVF3800-UA. The command to do that is the following:

```
(sudo) dfu-util(.exe) -d 0x20b1:0x4f00 -a 1 -D [UPGRADE_BINARY] -R
```

where `-a 1` is fixed and it indicates the altsetting for the upgrade image, and `UPGRADE_BINARY` is the path to the upgrade binary generated as described in *Generation of Binary Upgrade Image*. The `-R` option will trigger a reboot of the device after the download is complete.

Warning: If the download operation fails and the first data blocks have been transferred to the device, the device must be rebooted before another DFU download is performed. If this is not done, the host application incorrectly reports the last operation as successful. After the reboot the device will use the factory image, as any pre-existing upgrade image has been corrupted by the failed DFU operation.

7.3.2 Revert the device to factory image

To restore the device to its factory configuration, effectively discarding any upgrades made, the same process as outlined above is followed but using a blank upgrade image. This is the only way a restore can be initiated, as the device does not have the ability to restore itself.

The blank file can be generated using `dd` on MAC and Linux, and `fsutil` on Windows. A blank image can be created with a file of zeroes the size of one flash sector. In the normal case of 4KB sectors on a UNIX-compatible platform, this can be created as follows:

```
dd bs=4096 count=1 < /dev/zero 2>/dev/null blank.bin
```

and for Windows systems:

```
fsutil file createNew blank.bin 4096
```

This process outlined in the *DFU download operation* section can now be followed using *blank.bin* file as `UPGRADE_BINARY`.

7.3.3   Reboot the device

The *dfu-util* application can be used to reboot the device via the USB interface. This can be done with the following command:

```
(sudo) dfu-util(.exe) -d 0x20b1:0x4f00 -a 1 -e
```

in this command either `-a 0` and `-a 1` are valid.

7.3.4   Upload the images from the device

The DFU upload operation allows the user to read the factory and upgrade images of the XVF3800-UA. This feature is useful for debugging purposes, for example to provide XMOS with the exact firmware used on a device. The command to do that is the following:

```
(sudo) dfu-util(.exe) -d 0x20b1:0x4f00 -a [ALTSETTING] -U [IMAGE_BINARY]
```

where `ALTSETTING` is 0 for the factory image and 1 for the upgrade image, and `IMAGE_BINARY` is the path to the binary to store the image read from the device.

The uploaded upgrade image should match the same image created in *Generation of Binary Upgrade Image*, but without any extra suffix.

The uploaded factory image contains the bootloader data and the image obtained using the command:

```
xflash [FACTORY_EXECUTABLE] -o [OUTPUT_BINARY]
```

where `FACTORY_EXECUTABLE` is the path to the .xe application flashed on the device and `OUTPUT_BINARY` is the path to the binary to generate.

Note: All the images uploaded from the device have some padding bytes, so that their size is a multiple of DFU transfer size. This value is set to 256 bytes by default.

---

8 APPENDIX – Control Commands

There are a number of control commands available for use with the XVF3800. These are detailed using the `--list-commands` option of the Host Application, but may also be found below for reference. Note that all parameters in this appendix will be reset to their default values on a device reset.

8.1 AEC Tuning and Control Commands

These commands focus on tuning parameters for the AEC and postprocessing tasks.

Table 8.1: AEC and PP control commands

Command name	Read / Write	Params	Param format	Description
AEC_AECPATHCHANGE	READ ONLY	1	int32	AEC Path Change Detection. Valid range: 0,1 (false,true)
AEC_HPFONOFF	READ / WRITE	1	int32	High-pass Filter on microphone signals. Valid range: 0,1,2,3,4 (0:Off, 1:on70, 2:on125, 3:on150, 4:on180). Default value(s): on70
AEC_AECSILENCELEVEL	READ / WRITE	2	float	Power threshold for signal detection in adaptive filter.(set,cur), Valid range (set): [0.0 .. 1.0] Valid range (cur): 0.05*set, 1e-6f or set.. Default value(s): 1e-8f
AEC_AECCONVERGED	READ ONLY	1	int32	Flag indicating whether AEC is converged. Valid range: 0,1 (false,true). Default value(s): False
AEC_AECEMPHASISONOFF	READ / WRITE	1	int32	Pre-emphasis and de-emphasis filtering for AEC. Valid range: 0,1,2 (off,on,on_eq)on: Emphasis filter for speech signals without modification of far-end ref. on_eq: Emphasis filter for far-end reference signals where the low frequencies are boosted by e.g. an equalizer.. Default value(s): False
AEC_FAR_EXTGAIN	READ / WRITE	1	float	External gain in dB applied to the far-end reference signals. Valid range: [-inf .. inf]. Default value(s): 0.0dB
AEC_PCD_COUPLINGI	READ / WRITE	1	float	Sensitivity parameter for PCD. Valid range: [0.0 .. 1.0]PCD can be disabled by setting a value outside the range. Default value(s): disabled
AEC_PCD_MINTHR	READ / WRITE	1	float	Minimum threshold value used in PCD. Valid range: [0.0 .. 0.02]. Default value(s): 0.005
AEC_PCD_MAXTHR	READ / WRITE	1	float	Maximum threshold value used in PCD. Valid range: [0.025 .. 0.2]. Default value(s): 0.1
AEC_RT60	READ ONLY	1	float	Current RT60 estimate. Valid range: [0.250 .. 0.900] (seconds) A negative value indicates that the RT60 estimation is invalid.
SHF_BYPASS	READ / WRITE	1	uint8	AEC bypass

Table 8.1 – continued from previous page

Command name	Read / Write	Params	Param format	Description
AEC_NUM_MICS	READ ONLY	1	int32	Number of microphone inputs into the AEC
AEC_NUM_FARENDS	READ ONLY	1	int32	Number of farend inputs into the AEC
AEC_MIC_ARRAY_TYPE	READ ONLY	1	int32	Microphone array type (1 - linear, 2 - squarecular)
AEC_MIC_ARRAY_GEO	READ ONLY	12	float	Microphone array geometry. Each microphone is represented by 3 XYZ coordinates in cm
AEC_AZIMUTH_VALUES	READ ONLY	4	radians	Azimuth values in radians - beam 1, beam 2, free-running beam, 3 - auto-select beam
AEC_CURRENT_IDLE_TIME	READ ONLY	1	uint32	AEC processing current idle time in 10ns ticks
AEC_MIN_IDLE_TIME	READ ONLY	1	uint32	AEC processing minimum idle time in 10ns ticks
AEC_RESET_MIN_IDLE_TIME	WRITE ONLY	1	uint32	Reset the AEC minimum idle time. The value of the argument passed is ignored.
AEC_FILTER_CMD_ABORT	WRITE ONLY	1	int32	Reset the special command state machine. Used for safely exiting from an AEC filter read/write command sequence that has gone wrong.
PP_AGCONOFF	READ / WRITE	1	int32	Automatic Gain Control. Valid range: 0,1 (off,on). Default value(s): True
PP_AGCMAXGAIN	READ / WRITE	1	float	Maximum AGC gain factor. Valid range: [1.0 .. 1000.0] (linear gain factor). Default value(s): 31.6
PP_AGCDESIREDLEVEL	READ / WRITE	1	float	Target power level of the output signal. Valid range: [1e-8 .. 1.0] (power level). Default value(s): 0.0025
PP_AGCGAIN	READ / WRITE	1	float	Current AGC gain factor. Valid range: [1.0 .. 1000.0] (linear gain factor). Default value(s): 1.0
PP_AGCTIME	READ / WRITE	1	float	Ramp-up/down time-constant. Valid range: [0.5 .. 4.0] (seconds). Default value(s): 0.9
PP_AGCFASTTIME	READ / WRITE	1	float	Ramp down time-constant in case of peaks. Valid range: [0.05 .. 4.0] (seconds). Default value(s): 0.1f
PP_AGCALPHAFASTGAIN	READ / WRITE	1	float	Gain threshold enabling fast alpha mode. Valid range: [0.0 .. 1000.0] (linear gain factor). Default value(s): 0.0
PP_AGCALPHASLOW	READ / WRITE	1	float	Slow memory parameter for speech power. Valid range [0.0 .. 1.0]. Default value(s): 0.984
PP_AGCALPHAFAST	READ / WRITE	1	float	Fast memory parameter for speech power. Valid range [0.0 .. 1.0]. Default value(s): 0.36
PP_LIMITONOFF	READ / WRITE	1	int32	Limiter on communication output. Valid range: 0,1 (off,on). Default value(s): True
PP_LIMITPLIMIT	READ / WRITE	1	float	Maximum limiter power. Valid range: [1e-8 .. 1.0] (power level). Default value(s): 0.47

Table 8.1 – continued from previous page

Command name	Read / Write	Params	Param format	Description
PP_MIN_NS	READ / WRITE	1	float	Gain-floor for stationary noise suppression. Valid range: [0.0 .. 1.0]. Default value(s): 0.15
PP_MIN_NN	READ / WRITE	1	float	Gain-floor for non-stationary noise suppression. Valid range: [0.0 .. 1.0]. Default value(s): 0.15
PP_ECHOONOFF	READ / WRITE	1	int32	Echo suppression. Valid range: 0,1 (off,on). Default value(s): True
PP_GAMMA_E	READ / WRITE	1	float	Over-subtraction factor of echo (direct and early components). Valid range: [0.0 .. 2.0]. Default value(s): 1.0
PP_GAMMA_ETAIL	READ / WRITE	1	float	Over-subtraction factor of echo (tail components). Valid range: [0.0 .. 2.0]. Default value(s): 1.0
PP_GAMMA_ENL	READ / WRITE	1	float	Over-subtraction factor of non-linear echo. Valid range: [0.0 .. 5.0]. Default value(s): 1.0
PP_NLATTENONOFF	READ / WRITE	1	int32	Non-Linear echo attenuation. Valid range: 0,1 (off,on). Default value(s): 1
PP_NLAEC_MODE	READ / WRITE	1	int32	Non-Linear AEC training mode. Valid range: 0,1,2 (normal,train,train2). Default value(s): 0
PP_MGSCALE	READ / WRITE	3	float	Minimum gain scale for acoustic echo suppression.(max,min,cur), Valid range (max,min): [(1.0,0.0) .. (1e5,max)] Valid range (cur): min or max.. Default value(s): (1.0,1.0)
PP_FMIN_SPEINDEX	READ / WRITE	1	float	In case of double talk, frequencies below SPEINDEX are more suppressed than frequencies above SPEINDEX. The actual suppression is depending on the setting of DTSENSITIVE. The parameter is not a taste parameter but needs to be tuned for a specific device. Valid range: [0.0 .. 7999.0]. Default value(s): 593.75
PP_DTSENSITIVE	READ / WRITE	1	int32	Tradeoff between echo suppression and doubletalk performance. A lower value prefers high echo suppression (possibly at the cost of less doubletalk performance), a higher value prefers better doubletalk performance (possibly at the cost of good echo suppression). Good doubletalk performance is only possible for hardware without much non-linearities. When the value has 2 digits, for robustness an extra near-end speech detector is used. Valid range: [0 .. 5, 10 .. 15]. Default value(s): 12
PP_ATTNS_MODE	READ / WRITE	1	int32	Additional reduction of AGC gain during non-speech. Valid range: 0,1,2 (off, on, on with modified beamselection off). Default value(s): 0
PP_ATTNS_NOMINAL	READ / WRITE	1	float	Amount of additional reduction during non-speech at nominal speech level. Valid range: [0.0 .. 1.0]. Default value(s): 1.0

Table 8.1 – continued from previous page

Command name	Read / Write	Params	Param format	Description
PP_ATTNS_SLOPE	READ / WRITE	1	float	Determines the extra amount of suppression during non-speech when the AGC level increases at lower speech level. The extra attenuation is given by (agcgain_nominal/agcgain_current)^attns_slope. With a value of 1.0 the amount of noise in the output remains approximately the same, independent of the agc gain. Valid range: [0.0 .. 5.0]. Default value(s): 1.0
PP_CURRENT_IDLE_TIME	READ ONLY	1	uint32	PP processing current idle time in 10ns ticks
PP_MIN_IDLE_TIME	READ ONLY	1	uint32	PP processing minimum idle time in 10ns ticks
PP_RESET_MIN_IDLE_TIM E	WRITE ONLY	1	uint32	Reset the PP minimum idle time. The value of the argument passed is ignored.
PP_NL_MODEL_CMD_ABORT	WRITE ONLY	1	int32	Reset the special command state machine. Used for safely exiting from a NL model read/write sequence that has gone wrong.

8.2   Device Metadata Commands

These commands focus on querying the device's metadata, e.g. software version, boot status, and build information.

Table 8.2: Metadata control commands

Command name	Read / Write	Params	Param format	Description
VERSION	READ ONLY	3	uint8	The version number of the firmware. The format is VERSION_MAJOR VERSION_MINOR VERSION_PATCH
BLD_MSG	READ ONLY	50	char	Retrieve the build message built in to the firmware, normally the build configuration name
BLD_HOST	READ ONLY	30	char	Retrieve details of the CI build host used to build the firmware
BLD_REPO_HASH	READ ONLY	40	char	Retrieve the GIT hash of the sw_xvf3800 repo used to build the firmware
BLD_MODIFIED	READ ONLY	6	char	Show whether or not the current firmware repo has been modified from the official release. Requires use of a GIT repo
BOOT_STATUS	READ ONLY	3	char	Shows whether or not the firmware has been booted via SPI or JTAG/FLASH
TEST_CORE_BURN	READ / WRITE	1	uint8	Set to enable core burn to exercise worst case timing. This will reboot the chip, reset all parameters to default and significantly increase power consumption.
USB_BIT_DEPTH	READ / WRITE	2	uint8	Only relevant for the UA device variant. For the UA device, set or get the USB bit depth IN, OUT to either 16, 24 or 32. Setting will reboot the chip, resetting all other parameters to default. If issued to the INT device, a set is ignored and the device is not rebooted while a get always returns 0 as both IN and OUT bit depths.. Valid range: val0: [16 .. 16] val0: [24 .. 24] val0: [32 .. 32] val1: [16 .. 16] val1: [24 .. 24] val1: [32 .. 32]

8.3   Audio Manager Commands

These commands are targeted toward setting and retrieving various options around the audio path into and out of the device, including setting I²S loopback modes and debug "packed" IO modes. Includes diagnostic data on idle times for both the audio manager core and the I²S core.

Table 8.3: Audio Manager control commands

Command name	Read / Write	Params	Param format	Description
AUDIO_MGR_MIC_GAIN	READ / WRITE	1	float	Audio Mgr pre shf microphone gain. Default value(s): 80.0
AUDIO_MGR_REF_GAIN	READ / WRITE	1	float	Audio Mgr pre shf reference gain. Default value(s): 1.0
AUDIO_MGR_CURRENT_IDLE_TIME	READ ONLY	1	int32	Get audio manager current idle time in 10ns ticks
AUDIO_MGR_MIN_IDLE_TIME	READ ONLY	1	int32	Get audio manager min idle time in 10ns ticks
AUDIO_MGR_RESET_MIN_IDLE_TIME	WRITE ONLY	1	int32	Rest audio manager min idle time. The value of the argument passed is ignored.
MAX_CONTROL_TIME	READ ONLY	1	int32	Get audio manager max control time
RESET_MAX_CONTROL_TIME	WRITE ONLY	1	int32	Reset audio manager max control time. The value of the argument passed is ignored.
I2S_CURRENT_IDLE_TIME	READ ONLY	1	int32	Get I2S current idle time in 10ns ticks
I2S_MIN_IDLE_TIME	READ ONLY	1	int32	Get I2S min idle time in 10ns ticks
I2S_RESET_MIN_IDLE_TIME	WRITE ONLY	1	int32	I2S reset idle time. The value of the argument passed is ignored.
I2S_INPUT_PACKED	READ / WRITE	1	uint8	Will expect packed input on both I2S channels if this is not 0. Note, could take up to 3 samples to take effect. Valid range: val0: [0 .. 1]. Default value(s): 0
AUDIO_MGR_SELECTED_AZIMUTHS	READ ONLY	2	radians	The azimuths associated with the left and right channels of MUX_USER_CHOSEN_CHANNELS, how this aligns with actual output channels depends on the mux configuration
AUDIO_MGR_SELECTED_CHANNELS	READ / WRITE	2	uint8	Default implementation of post processing will use this to select which channels should be output to MUX_USER_CHOSEN_CHANNELS. Note that a customer implementation of the beam selection stage could override this command. How this channel selection aligns with actual output depends on the mux configuration. Valid range: val0: [0 .. 3] val1: [0 .. 3]. Default value(s): (3, 3)
AUDIO_MGR_OP_PACKED	READ / WRITE	2	uint8	<L>, <R>; Sets/gets packing status for L and R output channels. Valid range: val0: [0 .. 1] val1: [0 .. 1]. Default value(s): (0, 0)
AUDIO_MGR_OP_UPSAMPLE	READ / WRITE	2	uint8	<L>, <R>; Sets/gets upsample status for L and R output channels, where appropriate. Valid range: val0: [0 .. 1] val1: [0 .. 1]

Table 8.3 – continued from previous page

Command name	Read / Write	Params	Param format	Description
AUDIO_MGR_OP_L	READ / WRITE	2	uint8	<category>, <source>; Sets category and source for L output channel. Equivalent to AUDIO_MGR_OP_L_PK0. Valid range: val0: [0 .. 12] val1: [0 .. 5]. Default value(s): (8, 0)
AUDIO_MGR_OP_L_PK0	READ / WRITE	2	uint8	<category>, <source>; Sets category and source for first (of three) sources on the L channel in packed mode. Equivalent to AUDIO_MGR_OP_L. Valid range: val0: [0 .. 12] val1: [0 .. 5]
AUDIO_MGR_OP_L_PK1	READ / WRITE	2	uint8	Sets category and source for second (of three) sources on the L channel in packed mode. Valid range: val0: [0 .. 12] val1: [0 .. 5]. Default value(s): (1, 0)
AUDIO_MGR_OP_L_PK2	READ / WRITE	2	uint8	Sets category and source for third (of three) sources on the L channel in packed mode. Valid range: val0: [0 .. 12] val1: [0 .. 5]. Default value(s): (1, 2)
AUDIO_MGR_OP_R	READ / WRITE	2	uint8	<category>, <source>; Sets category and source for R output channel. Equivalent to AUDIO_MGR_OP_R_PK0. Valid range: val0: [0 .. 12] val1: [0 .. 5]. Default value(s): (3, 2)
AUDIO_MGR_OP_R_PK0	READ / WRITE	2	uint8	<category>, <source>; Sets category and source for first (of three) sources on the R channel in packed mode. Equivalent to AUDIO_MGR_OP_R. Valid range: val0: [0 .. 12] val1: [0 .. 5]
AUDIO_MGR_OP_R_PK1	READ / WRITE	2	uint8	Sets category and source for second (of three) sources on the R channel in packed mode. Valid range: val0: [0 .. 12] val1: [0 .. 5]. Default value(s): (1, 1)
AUDIO_MGR_OP_R_PK2	READ / WRITE	2	uint8	Sets category and source for third (of three) sources on the R channel in packed mode. Valid range: val0: [0 .. 12] val1: [0 .. 5]. Default value(s): (1, 3)
AUDIO_MGR_OP_ALL	READ / WRITE	12	uint8	Sets category and source for all 3 sources on L channel and all 3 sources for R channel. Equivalent to AUDIO_MGR_OP_[L,R]_PK[0,1,2] with successive pairs of arguments. Valid range: val0: [0 .. 12] val1: [0 .. 5] val2: [0 .. 12] val3: [0 .. 5] val4: [0 .. 12] val5: [0 .. 5] val6: [0 .. 12] val7: [0 .. 5] val8: [0 .. 12] val9: [0 .. 5] val10: [0 .. 12] val11: [0 .. 5]
I2S_INACTIVE	READ ONLY	1	uint8	Returns whether the main audio loop is exchanging samples with I2S (0). If not (1), I2S is inactive
AUDIO_MGR_FAR_END_DSP_ENABLE	READ / WRITE	1	uint8	Enables/disables far-end DSP (if implemented). Write a 1 to enable, 0 to disable. Valid range: val0: [0 .. 1]. Default value(s): 0
AUDIO_MGR_SYS_DELAY	READ / WRITE	1	int32	Delay, measured in samples, that is applied to the reference signal before passing to SHF BeClear algorithm. Valid range: val0: [-64 .. 256]. Default value(s): 0

XMOS XVF3800 - User Guide

8.4   GPIO Commands

These commands set up and manipulate various functions of the device's GPO and GPI services.

Table 8.4: GPIO control commands

Command name	Read / Write	Params	Param format	Description
GPI_INDEX	READ / WRITE	1	uint8	Set/get pin index for next and subsequent GPI reads. Maximum value should be equal to GPIO_NUM_INPUT_PINS. Valid range: val0: [0 .. 1]
GPI_EVENT_CONFIG	READ / WRITE	1	uint8	Set/get event config for selected pin. Valid range: val0: [0 .. 3]
GPI_ACTIVE_LEVEL	READ / WRITE	1	uint8	Set/get active level for selected pin
GPI_VALUE	READ ONLY	1	uint8	Get current logic level of selected GPI pin.
GPI_EVENT_PENDING	READ ONLY	1	uint8	Get whether event was triggered for selected GPI pin. Event flag is cleared for the pin. Interrupt pin is set when all event flags are cleared
GPI_VALUE_ALL	READ ONLY	1	uint32	Get current logic level of all GPI pins as a bitmap, where GPI index n -> bit n of returned value.
GPI_EVENT_PENDING_ALL	READ ONLY	1	uint32	Get whether event was triggered for all GPI pins as a bitmap, where GPI index n -> bit n of returned value. Event flag is cleared for all GPI pins. Interrupt pin is set.
GPO_PORT_PIN_INDEX	READ / WRITE	2	uint32	GPO port index and pin index that the following commands would be directed to. Valid range: val0: [0 .. 0] val1: [3 .. 7]
GPO_PIN_VAL	WRITE ONLY	3	uint8	value to write to one pin of a GPO port. Payload specifies port_index, pin_index and value to write to the pin. Valid range: val0: [0 .. 0] val1: [3 .. 7] val2: [0 .. 1]
GPO_PIN_ACTIVE_LEVEL	READ / WRITE	1	uint32	Active level of the port/pin specified by the GPO_PORT_PIN_INDEX command. 1 = ACTIVE_HIGH, 0 = ACTIVE_LOW. Valid range: val0: [0 .. 1]
GPO_PIN_PWM_DUTY	READ / WRITE	1	uint8	PWM duty cycle of the pin specified by the GPO_PORT_PIN_INDEX command. Specified as an integer percentage between 0 and 100. Valid range: val0: [0 .. 100]
GPO_PIN_FLASH_MASK	READ / WRITE	1	uint32	Serial flash mask for the pin specified by the GPO_PORT_PIN_INDEX command. Each bit in the mask specifies the GPO pin state for a 100 ms time period interval