AN14276

Voice Seeker and VoiceSpot for i.MX

Rev. 1.0 — 24 June 2024 Application note

Document information

Information	Content
Keywords	AN14276, Voice Processing, VoiceSeeker, VoiceSpot
Abstract	NXP created the Voice Processing environment, which is continuously improved. This environment enables customers to use it for their application with low latency, low false-positives, low energy consumption, and with minimum effort.

1 Introduction

NXP created the Voice Processing environment, which is continuously improved. This environment enables customers to use it for their application with low latency, low false-positives, low energy consumption, and with minimum effort.

VoiceSeeker and VoiceSpot are some of the tools that NXP offers for voice processing. They can be used with most of the i.MX 8M family and with low CPU usage.

2 Purpose

This document introduces VoiceSeeker, a library providing high-resolution beamforming and multichannel Acoustic Echo Cancellation (AEC), without requiring a predetermined fixed microphone geometry, which translates to more flexibility on board design.

Alongside VoiceSeeker, this document describes VoiceSpot, a speech-detection engine which does not require to be recompiled to work with different models. It fits perfectly with VoiceSeeker, because both of them were designed to work together. VoiceSeeker removes the noise from the audio. When it is done, VoiceSeeker sends the buffer to VoiceSpot, which only focuses on detecting the wake word. If a wake word is detected, VoiceSpot sends feedback back to VoiceSeeker.

The main objectives of this document are:

- To fully understand what VoiceSeeker, VoiceSpot, and AFE are.
- To configure VoiceSeeker and VoiceSpot as well as their dependencies.
- To distinguish the difference between the default and external configurations.
- To understand the calibration process.

3 Overview

The first stage of a voice assistance is having the ability to listen when spoken to. Despite being the first stage of voice assistance, it is very important since the buffer used to trigger the voice assistance is the same buffer used in the following steps. This means that the buffer should be cleaned and have only the human artifacts to have a good accuracy of the wake-word detection and the interpretation of the commands. The main goal of VoiceSeeker is to clean up the input microphone data while VoiceSpot is the mechanism used to detect the wake word and trigger the next stages of voice assistance.

VoiceSeeker is a library that processes the microphone input. One of these processes is to remove unwanted noise from the data captured by the microphones. The core of VoiceSeeker is a static library, which has the APIs to process the audio. Imx-voiceui project takes that static library and adds a new layer, which makes it easier to work with. The addition of this layer with the static library gives a shared object as the result. From now on, when talking about VoiceSeeker, it will be a reference to this shared object.

The VoiceSeeker library comes preinstalled on most of the NXP Linux distributions, so you can have a quick taste of some of its capabilities. VoiceSeeker is a library, which means that it requires an application that can load it at runtime. The application that can load it is called AFE and it also comes preinstalled.

AFE is a program which allows to select the desire engine for voice processing at runtime; for example, choosing either VoiceSeeker or Conversa. An overview of AFE is provided in the following chapters and focused on the use case, where it loads the VoiceSeeker library. For more use cases of the AFE, see the TODO.md file on GitHub.

The `voice_ui_app` is another program from NXP's Voice Processing Environment. The `voice_ui_app` is waiting for the output of VoiceSeeker to search if there are any voice artifacts. Voice artifacts can be either the wake word or a voice command and each of them is handled in a different way. First, it looks for the wake word

with the help of VoiceSpot and if VoiceSpot finds the wake word command, it has the ability to trigger a third-party application so the third party starts listening to the capture stream. VoiceSpot also sends the buffer to VIT so it can look for the voice commands. In the scenario where VoiceSpot could not find the wake word it would only send the buffer to VIT without sending any type of signal to the third-party application. This document only explains the workflow from the microphone capture using AFE with VoiceSeeker until the detection of the wake word with VoiceSpot in `voice_ui_app`. For more detailed information about the VoiceSpot API, see its documentation.

VIT (Voice Intelligent Technology) is another NXP product to handle voice commands. It is not in the scope of this document, but it can be integrated with VoiceSeeker and VoiceSpot (see Section 5.1.4). If you want to learn more about VIT, visit its website or see *Getting Started with VIT for i.MX RT devices: Voice Intelligent Technology SDK demo* (document IMXRTVITGSUG).

4 Materials

This section describes the hardware and software requirements needed to properly execute the example shown in this document.

4.1 Software requirements

The following software is needed to run the example described in this document:

- i.MX Linux BSP version 6.6.23 or higher (the recommended version or any other version can be downloaded from https://www.nxp.com/IMXLINUX).
- Audacity or a similar software.

For more information about how to flash the board and get it ready to run the BSP, see the *i.MX Linux User's Guide* (document IMXLUG).

4.2 Hardware requirements

The following hardware is needed to run the example described in this document:

- i.MX 8MP EVK board.
- 8MIC-RPI-MX8 board.
- Speakers with AUX connection.
- USB-A to USB-C cable.
- USB-A to USB-B micro cable.
- 3.5 mm jack audio cable.

If you have any concerns about the hardware setup, the i.MX 8M Plus EVK Quick Start Guide explains how to download the software and boot the board.

5 Voice-UI framework

This section explains the Voice-UI framework.

5.1 Integration to NXP Linux BSP

All programs and binaries mentioned in this document come with each BSP release. However, some of those libraries or programs are for evaluation only. For example, VoiceSeeker only comes with microphone beamforming features, which allow to detect wake words using VoiceSpot. The full version of VoiceSeeker comes with the AEC (Acoustic Echo Cancellation) and DOA (Direction Of Arrival) features. These features are

useful when developing voice applications. If you are interested in testing the full version, send an email to voice@nxp.com to receive further information about this.

5.1.1 Architecture

VoiceSeeker and VoiceSpot were designed to work independently from each other. However, it is recommended to use them together to achieve the best behavior of each library. VoiceSeeker requires feedback that indicates that a wake word was activated so that it can be adjusted based on this information. This feedback is given by VoiceSpot in the exact moment that it detects a wake word. If VoiceSeeker is working alone, you may want to implement such a mechanism to reach the best performance on the beamforming. The reason to do so is because each library is running on separate processes and VoiceSeeker needs the feedback to work correctly. However, if they are running together, this mechanism is already implemented in the libraries they are wrapped on.

The binaries that come in the Linux release demonstrate a use case where a small voice assistance can process certain number of commands locally. This small assistance does not depend on the Internet connection to work. The cleanup of the signal is done by VoiceSeeker. VoiceSpot searches for the wake word and VIT decodes and processes the voice command. The VIT library also runs in `voice_ui_app`.

The architecture of the full use case described in this document is shown in Figure 1.

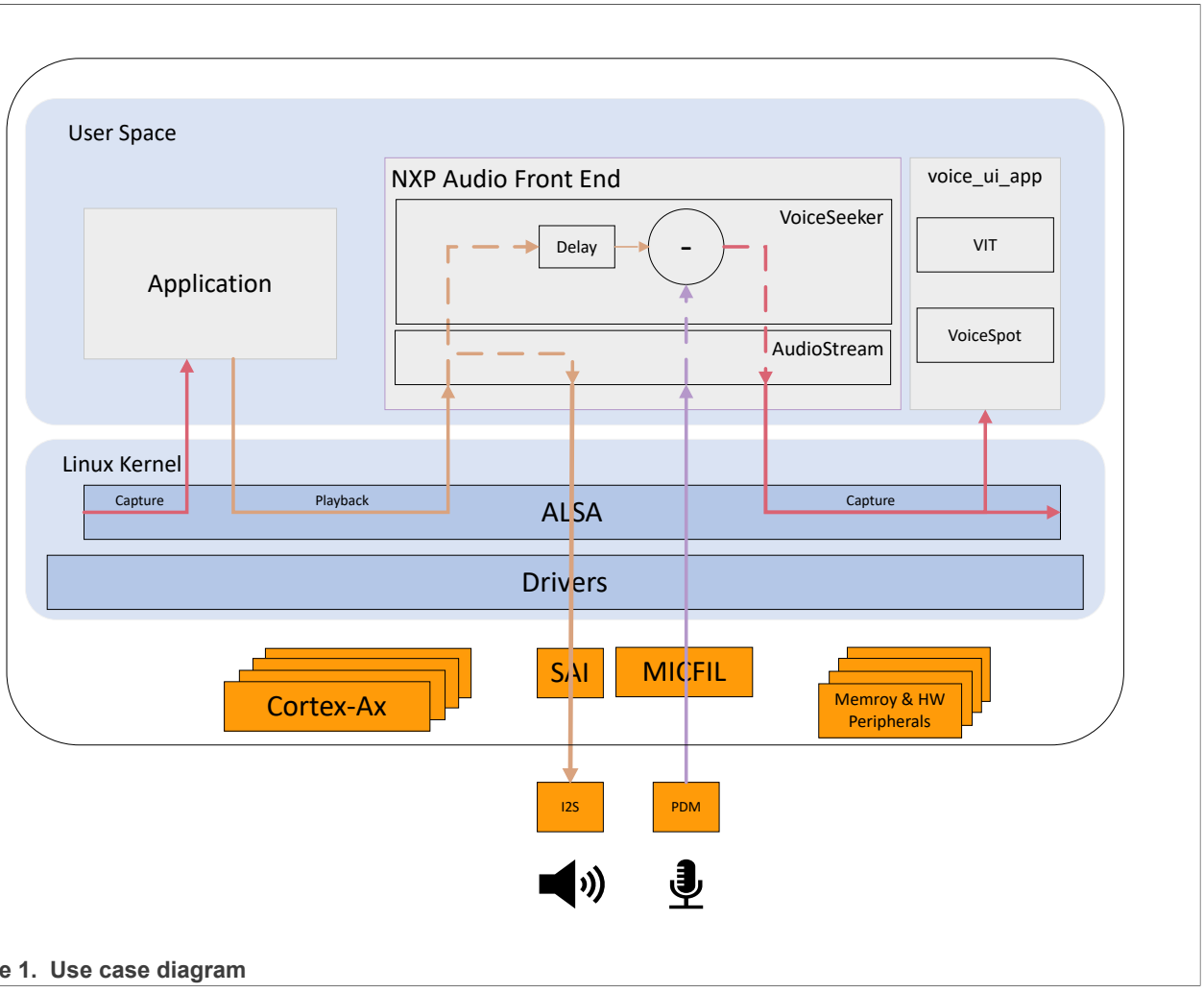

Figure 1. Use case diagram

Another independent application, which must use the default ALSA device, is responsible for starting the playback stream so that AFE can manage the incoming and outgoing data. It is assumed that the `asound.conf` file has a configuration compatible with AFE. Therefore, `voice_ui_app` must be started before anything else with the VoiceSpot and VIT libraries. AFE must be started after it. AFE loads VoiceSeeker dynamically. Finally, a playback stream can be started. On the other side, if the `asound.conf` file does not have a compatible configuration (AFE will not be able to open its devices), AFE issues an error and stops its execution. The compatible `asound.conf` file is described later on in the document.

When VoiceSeeker (which is running on AFE) finishes its process, VoiceSpot is notified to look for a wake word in the VoiceSeeker output, a buffer with only human voice artifacts. Finally, if a wake word is found, VoiceSpot can notify a third application for the processing of the voice command and VoiceSpot gives the feedback to VoiceSeeker to adjust the beamforming. Since VoiceSpot is the last process in this flow, it must be the first to initialize, even before AFE. The initialization process of this use case or any other similar use case is as follows:

1. Start the VoiceSpot shared library with `voice_ui_app` or with other software.
2. Start the AFE with VoiceSeeker for AEC and beamforming (only after `voice_ui_app` finishes its initialization).
3. Run the upper application (any application that plays audio; for example, `aplay`, `gst_launche`, Alexa).

5.1.1.1 External references signal

The architecture described above is useful when there is no extra postprocessing beyond the AFE (the data handled by AFE is the same data played on the speakers) or when it is not a real-time system, where meeting a deadline is critical. In those scenarios, the architecture of the system may require some changes. AFE can handle those scenarios as well.

The initialization process is similar to the above use case, unless you use an RTOS for real-time process. In this scenario, be sure that the RTOS is up and running and exposing a sound card on the Linux side. On Linux, run VoiceSpot and AFE, as mentioned above. This is a multicore application where it is required to have a real-time OS managing the playback and another OS to process the voice commands.

Both scenarios are described in this document (loopback and external modes). An external configuration is when the audio played is not generated in the user space. Instead, the audio source comes from a sound card and not from an application in the user space.

5.1.2 NXP AFE

The NXP AFE is an audio-stream managing module for the i.MX Linux BSP. It was designed so it could load and execute a given voice algorithm (selected as a command-line argument). It manages the connection between ALSA audio cards and the algorithm’s input and output buffers.

AFE was created as a solution for voice-processing libraries, where you must control the audio inputs and outputs of the system. In the particular case of VoiceSeeker, the output signal (the one heard through the speakers) must be stored as a reference to filter out noise. The incoming signal (the one captured by the microphones which have noise, references, and human voice) must be cleaned up before going to a deeper process. Both signals are used by VoiceSeeker and a third buffer, which only contains the human voices coming from the microphones, is generated.

AFE removes the stream control out of the voice-processing stage. This means that you can have a processing stage that only manipulates the buffers and leaves the audio flow to AFE. AFE calls these processes whenever the required buffers are ready to be processed. In the case of VoiceSeeker, whenever the microphones' and references' buffers are ready.

Finally, AFE handles the output buffer from the voice process and sends it to the default ALSA device, where any other client can be listening and waiting for a clean voice signal. Some examples of clients include VoiceSpot, VIT, MS Teams, Alexa, and others.

AFE only controls the streams. It does not even fix the delay between the streams (acoustic delay). Keep this in mind, because you might require a mechanism to solve such a delay (the solution and explanation of the acoustic delay are described further on in this document).

AFE on its own does not do any audio processing. It needs an additional library to process the signals. It can load any library that uses its APIs and it is located in the `/usr/lib/nxp-afe` folder. If the library follows these two rules, AFE can load it and work with it. To tell AFE which engine to use is done by providing its name as an argument (just the name, not the full path or extension). For more information on how to use AFE with different libraries, see the TODO.md file.

5.1.2.1 asound.conf

AFE relies on the asound.conf file. In this file, the ALSA (Advanced Linux Sound Architecture) devices are defined and can be opened through the ALSA APIs (Application Programming Interfaces). AFE uses those APIs to control the data flow feeding the voice engine and the ALSA default capture device.

The i.MX Linux BSP contains `asound` files for each board. The files are located in `/unit_tests/nxp-afe/asound.conf_<platform>` and can be customized when the hardware is different. There is a different `asound.conf` file for each board, because each board uses a different playback codec.

On each file, there are six main devices. Four of those devices are controlled by AFE and the last two of them are the main default playback and capture devices, where the application should write to (`pwloop`) and read from (`crloop`). The audio that must be heard through the speakers should write to `pwloop`. To capture the voice artifacts, the application must use `crloop`. The other devices are managed by AFE. Two of them are the actual physical source and sink-audio devices. One plays through the speaker (`spk`) and the other captures the input of the microphones (`mics`). The last two are virtual devices. One acts as a sink, while the other one acts as a source. `prloop` is a sink that captures the audio that the application is trying to play. It copies the buffer and writes it to the speakers. Similarly, `cwloop` is a virtual device that works as a source for the application, trying to capture the voice artifacts. AFE passes this buffer to VoiceSeeker, which uses this buffer to remove the data coming from the speakers (using the copy it just made of the speaker data). When the process finishes, it returns a new buffer with the clean audio. AFE propagates the data to the default capture device (`crloop`) through `cwloop`.

Figure 2 illustrates the data flow through AFE:

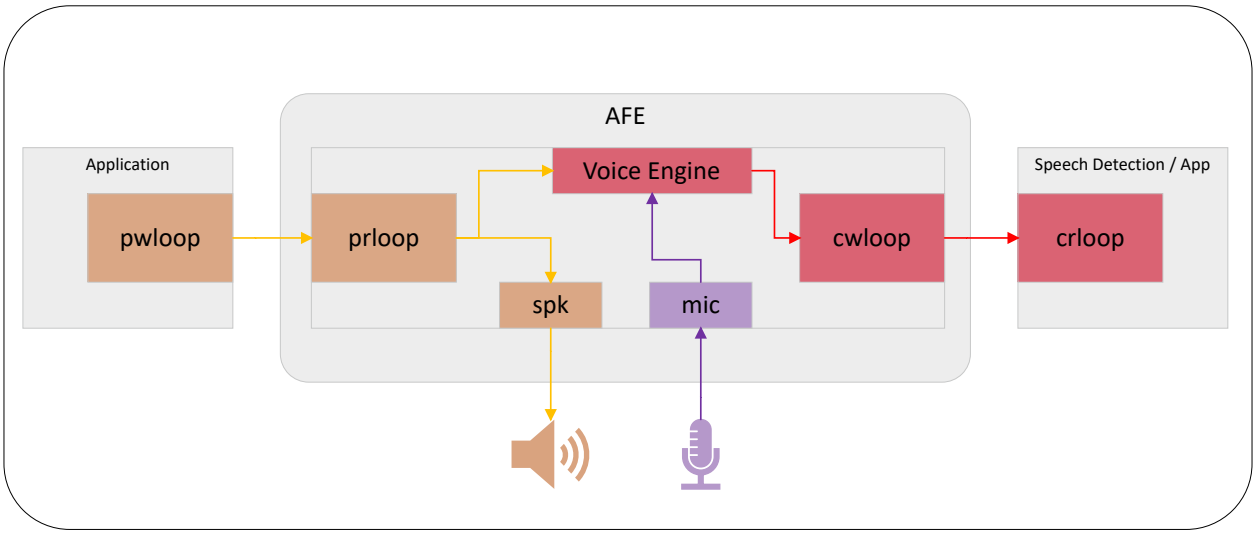

Figure 2. Data flow through AFE

Table 1 shows the properties of the six devices.

Table 1. Device properties

Device name	Owner	Access	Type	Pipeline
pwloop	Application	Write	Loopback	Playback
prloop	AFE	Read	Loopback	Playback
spk	AFE	Write	Hardware	Playback
mic	AFE	Read	Hardware	Capture
cwloop	AFE	Write	Loopback	Capture
crloop	Application	Read	Loopback	Capture

5.1.2.1.1 External references signal

In the external references configuration, AFE disables the `spk` device, because it is not responsible of playing the audio and the `prloop` is not opened. Instead, it opens another device, which must have the same data that is being played on the speakers. The capture pipeline remains the same.

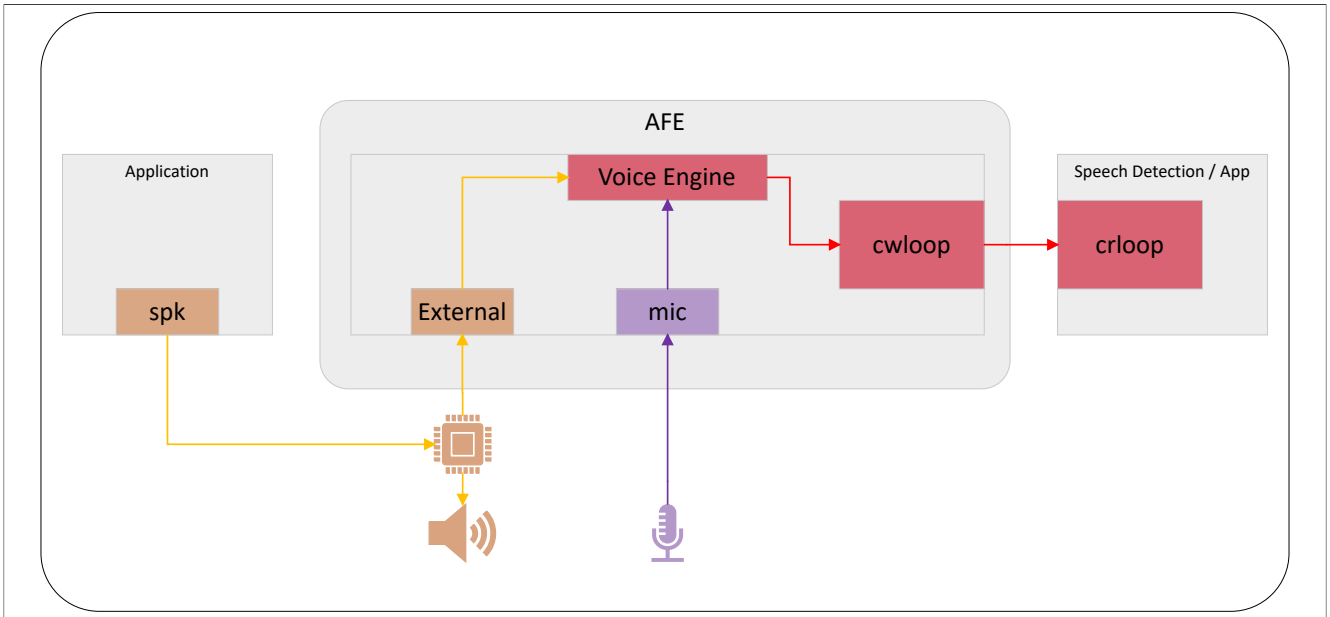

Figure 3. External references configuration AFE

5.1.3 VoiceSeeker

VoiceSeeker is NXP's voice-processing library used to clean up the incoming data. The full name of the solution is VoiceSeeker Light (VSL). VoiceSeeker can do the following:

- Detect the microphone from which the spoken word comes from and focus on that microphone (beamforming).
- Remove the ambient background noise and give a clean version of the word being spoken (AEC).
- Adapt to any microphone array, regardless of its physical location.
- Fix the acoustic delay (see Section 8.1).

VoiceSeeker is an example of a library that uses AFE APIs, has all its methods, and is built as a share object, allowing AFE to load it (if required). AFE can load VSL using the `afe libvoiceseekerlight` command. When the AFE loads the VoiceSeeker library, it sets up the ALSA devices according to the library requirements. The default requirements from VoiceSeeker are the following:

- Four channels for microphone input.
- Two channels for speaker output.
- 16000-kHz sample rate on all devices.
- `S32_LE` format on all devices.

When all the devices are opened and configured, AFE initializes the respective ALSA devices and starts streaming. AFE controls the streams, and when there is enough data to process, the VSL process is called. Inside the VSL process, the delay is fixed before processing the data and the audio is cleaned. When VSL finishes processing the data, it writes the data to the output buffer for which AFE is responsible to write to the loopback interface where the rest of the applications are listening.

This is a brief explanation of what VSL does inside AFE. However, there are important parts of the process that you should be aware of and configure. Starting from the beginning, on VSL initialization, there are a few things going on. The first one is the initialization of some variables, which depends on the number of channels to be processed. If you want to use more channels than the default ones (four microphones and two references signals) you must change the `heap_size` and the `scratch_size`. Another variable that depends on the number of reference channels is `sizeBufDelay`, which also depends on the amount of acoustic delay on the system. If you have more reference signals, or microphones too far from the speakers, change this variable.

Continuing with the initialization, VSL can be configured using the `vsl_config` variable, which is a structure that contains the elements listed in Table 2:

Table 2. variable vsl_config elements

Element	Description
num_mics	Number of microphones
num_spks	Number of speakers (references)
framesize_out	Output buffer size
buffer_length_sec	Time of the expected wake word
aec_filter_length_ms	Window size
create_aec	Enable AEC
create_doa	Enables DOA (Direction of Arrival)
mic_xyz_mm	Distance between microphones
device_id	Platform ID for i.MX 8M or i.MX 9, listed here

The first two variables are self-explanatory, they just hold the amount of reference and microphones. The third one is a little bit tricky. VoiceSeeker processes chunks of 32 frames per iteration and it returns a pointer to the filtered buffer when the buffer length has the number of samples requested. This number of samples is set using the `framesize_out` file, which is 200 samples. It is the default and recommended value due to the VoiceSpot limitation. The fourth element is related to the wake word size. For example, if you use a long word as a wake word (like “Hey NXP”), increase this value up to three seconds. If it is shorter (like “Alexa”), the value of this element can be set to 1.5 seconds (which is the default value).

`aec_filter_length_ms` is the window size. With a higher value, it takes longer on each iteration and it is not effective. Keeping it between 150 ms and 300 ms is a good choice.

The next variable enables the AEC algorithm. The free version does not allow to enable AEC. To get the full version, contact our team at Voice@nxp.com. This document covers the free and nonfree version of the library. The other thing that you can enable is the Direction of Arrival (DOA), but this feature is disabled by default on the preinstalled version and not covered in this document.

`mic_xyz_mm` is a matrix with the size of the number of channels times three dimensions (width, depth, and height). The distance between an origin and the microphones is measured in millimeters. The origin can be anywhere, not necessarily in one of the microphones.

VSL must know where it is running, because it has some optimization depending on which platform it is running on and that is what the last variable is for.

The last thing about the constructor function is that the mechanism, used to fix the acoustic delay, is created and initialized. This mechanism needs calibration before it is used to achieve a reliable performance. The calibration method is described in Section 8.1.

5.1.3.1 Config.ini

VoiceSeeker reads a file to be properly configured. This file is called `Config.ini` and it is located in the `/unit_tests/nxp-afe/` folder. During initialization, VoiceSeeker, as well as VoiceSpot, reads and parses the file to configure itself.

The parameters of the files and what they do is described in Table 3.

Table 3. File parameters

Field	Type	Description	Options
WWDectionDisable	Int	Disables the wake word detection in VoiceSpot or VIT.	0 or 1
WakeWordEngine	String	The engine used for the wake word.	VoiceSpot or VIT
DebugEnable	Int	Enables the dump of the buffer into a file. Required for the library calibration.	0 or 1
RefSignalDelay	Int	Value of the existing acoustic delay in samples.	[0, sizeBuffDelay - 1]
mic0	Float	Distance between the origin at the first microphone in milliseconds.	Float
mic1	Float	Distance between the origin at the second microphone in milliseconds.	Float
mic2	Float	Distance between the origin at the third microphone in milliseconds.	Float
mic3	Float	Measure between the origin at the fourth microphone in milliseconds.	Float
VoiceSpotModel	File	Model file to use within VoiceSpot. This is a field for VoiceSpot. VoiceSpot looks for the file in the same path where the `Config.ini` file is located.	String
VoiceSpotParams	File	Model parameters file. This is a field for Voice Spot. VoiceSpot looks for the file in the same path where the `Config.ini` file is located.	String
VITLanguage	String	Language of the VIT model	Please visit VIT for the available languages.

VoiceSeeker uses this file to configure itself. If the file is not present, VoiceSeeker falls back to the default values. The default values are the same as those listed in the file. If a field is not present in the configuration file, VoiceSeeker sets up with the default one, so make sure that there are no typos in the fields.

5.1.4 VoiceSpot

As well as VoiceSeeker, VoiceSpot is a library that is wrapped in a class (called `SignalProcessor_VoiceSpot`) for easy use of the voice engine. VoiceSpot is an independent program that can communicate to VoiceSeeker to retrieve the feedback of where and when a voice signal is detected. This is why VoiceSeeker and VoiceSpot work together perfectly. VoiceSeeker notifies when there is a chunk of data available and VoiceSpot tells VoiceSeeker if there are voice artifacts in that incoming data, so VoiceSeeker can focus its attention on that channel and readjust its filter on the run.

Just as VoiceSeeker, VoiceSpot is an object. There must be a main process that controls the workflow of the data and creates an instance of the library. This program is called `voice_ui_app`. It was previously called `voicespot`. The program is responsible for opening the capture ALSA device. When the data is ready, it calls either VoiceSpot or VIT for the voice decoding. It notifies (if configured to do so) a third application so that it can start an action based on the command that just arrived or on the wake word. Such applications can be voice assistance (or similar applications).

`Voice_ui_app` works with VIT for the wake-word detection and command detection or just the command detection. In the first scenario, VoiceSpot is completely disabled and VIT is responsible for notifying VoiceSeeker when a wake word or a command is detected. When VoiceSpot is enabled, `voice_ui_app` calls VoiceSpot to search for the wake word. If VoiceSpot is able to detect the wake word, then it passes the buffer to VIT to process the incoming voice command. To set up VIT for the wake-word engine, edit the configuration file. Set the `WakeWordEngine` variable to VIT instead of VoiceSpot.

The notification mechanism is enabled by passing the `-notify` flag when running `voice_ui_app`. If the flag is passed, both engines notify a third application. This means that VoiceSpot notifies when there is a wake word on the buffer and VIT notifies the ID of the voice command detected. When VoiceSpot is disabled, VIT still notifies when a wake word is detected.

5.1.4.1 Model

VoiceSpot uses a Machine Learning (ML) model, which had been trained with the best qualities and environment to achieve great precision even in a nosy environment, to detect the wake word. Therefore, if there is an interest in using VoiceSpot with a different model, contact us at Voice@nxp.com to get a better idea of the costs and time it takes to train the model for your custom wake word.

When you have your model and the parameters of the model, make sure that those files are in the `/unit_tests/nxp-afe` folder, so that VoiceSpot can find them.

6 Installation guide

This section describes the installation.

6.1 Hardware checklist

This section describes the hardware checklist.

6.1.1 For i.MX 8M Plus

VoiceSeeker and VoiceSpot are currently supported by NXP i.MX ArmV8-A processors, such as i.MX 8M Plus and i.MX 8M MINI, and it supports the ArmV8.2 architecture, such as i.MX 93. The i.MX 8M Plus EVK packages (as well as the other packages) contain the following items to evaluate VoiceSeeker and VoiceSpot:

Table 4. Items to evaluate VoiceSeeker and VoiceSpot

Quantity	Name	Description
1	i.MX 8M Plus EVK board	NXP evaluation board for the i.MX 8M Plus.
1	Power supply	USB-Type C 45-W power delivery supply, 5 V/3 A; 9 V/3 A; 15 V/3 A; 20 V/2.25 A supported.
1	Type-C male to type-A male cable	Assembly, USB 3.0 compliant.
1	Type-A male to micro-B male cable	Assembly, USB 2.0 for UART debug.

Additional hardware is also required:

Table 5. Additional hardware

Quantity	Name	Description
1	8MIC-RPI-MX8	NXP 8 microphone board. Available at https://www.nxp.com/part/8MIC-RPI-MX8#/.
1	Speakers	A speaker or two speakers compatible with a 3.5-mm audio jack connection.
1	3.5-mm cable	A 3.5-mm audio connection cable which connects the EVK and the speakers.

6.2 Setup view

This section describes a common setup to test the VoiceSeeker and VoiceSpot libraries. Each setup can vary but the connection must be the same.

Figure 4. Common setup

1. For the 8MIC-RPI-IMX8 board:
   a. Mount the board onto the `EXP_CN` of the EVK. Make sure that its pin numbering matches that on the EVK.
   b. Unmute the board by pulling down its mute switch. The switch indicates the muting when the board is powered on if this is not set.
   c. Pull up the `MIC_SEL` so that the ON position is active.
2. For the 8M Plus:
   a. Connect the audio jack table to the EVK and to the speakers.
   b. Connect the power supply.
   c. Connect the debug table to a PC.

6.3 Software installation

This section describes all the software, programs, and files that are required by VoiceSeeker and VoiceSpot.

6.3.1 Software checklist

The recommended BSP version is MM_04.09.00_2405_lf6.6.23 or higher. The latest image can be installed from https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/embedded-linux-for-i-mx-applications-processors:IMXLINUX.

When the image is fully downloaded, you can install it by following the steps in the *i.MX Linux User's Guide* (document IMXLUG).

The image should contain the files and drivers shown in Table 6.

Table 6. Files and drivers

File	Location	Description
libvoiceseekerlight.so	/usr/lib/nxp-afe/	VoiceSeeker wrapper. Loaded by AFE at runtime.
afe	/unit_tests/nxp-afe/	AFE binary.
voice_ui_app	/unit_tests/nxp-afe/	Main application.
asound.conf_imx8mp	/unit_tests/nxp-afe/	ALSA configuration file.
Confing.ini	/unit_tests/nxp-afe/	VoiceSeeker and VoiceSpot configuration file.
HeyNXP_1_params.bin	/unit_tests/nxp-afe/	NXP parameters for wake word.
HeyNXP_en-US_1.bin	/unit_tests/nxp-afe/	NXP wake-word model.
snd-aloop	/lib/modules/<bs_version> /kernel/sound/drivers	Driver for creating loopback devices inside ALSA.

7 Software setup

This section describes the software setup.

7.1 Selecting the device tree for the 8MIC-RPI-MX8 board

After flashing the Linux BSP to the desired storage medium of the EVK, boot the board and stop at the U-Boot terminal. This can be done by pressing any key after the U-Boot prompt during boot. When the U-Boot console is ready, enter the following commands:

```
u-boot=> editenv fdtfile
edit: imx8mp-evk-<revision>-8mic-revE.dtb
u-boot=> saveenv
u-boot=> boot
```

The given device trees for the revisions are shown in Table 7:

Table 7. Device trees

Device tree	Supported i.MX 8MP EVK revision
Imx8mp-evk-revb4-8mic-revE.dtb	B3, B4
Imx8mp-evk-revA3-8mic-revE.dtb	A3, B, B1
Imx8mp-evk.dtb	A2 or older

7.2 Installing missing drivers

By default, the image does not load the drivers that allow ALSA to do a loopback stream. Therefore, it is required to manually install it using the following command:

```
$ modprobe snd-aloop
```

7.3 Editing the ALSA asound configuration for VoiceSeeker

The following commands overwrite the ALSA virtual devices to properly work with VoiceSeeker while saving a backup of the original configuration.

```
$ cd /unit_tests/nxp-afe/
$ mv /etc/asound.conf{,.back}
$ cp asound.conf_imx8mp /etc/asound.conf
```

Copy the `asound.conf_imx8mp_revb4` file if the EVK revision is B3 or B4.

8 Audio lab guides

This section describes how to put the library in shape, as well as ways to test the library.

8.1 Delay measurement

As mentioned in the introduction, the AFE performs AEC (by loading the VoiceSeeker library) by subtracting the playback signal from the capture signal (which is a mix of the playback and voice signals). But the playback signal that comes in the capture stream mixed with voice artifacts has some delay with respect to the original playback because of the time that takes the sound to travel from the speakers to the microphones and be recorded, so it is important to have a precise measurement of this delay to achieve a good performance.

Before measuring the delay, it is necessary to mention a few things:

1. This is a one-time calibration, as long as the setup does not change. If the distance between microphones and speakers changes, it requires a recalibration.
2. The delay varies between measurements, but the range between measurements should be around one or two samples.
3. There are other ways to measure the delay. For example, another method is to use a cross-correlation between the L/R (Left and Right) speakers, and the microphone channels. You can use MATLAB, Python, or any other language that has a library that does that.
4. The method used in this document is a more visual method with an audio-tuning application, which counts the samples between two points in the analyzed audio stream.

To measure the delay, it is necessary to make a sample recording using AFE. In this sample recording, AFE generates three audio files: the playback + capture stream, the playback stream alone, and the capture stream alone (the result of AEC, which is not needed for calibration purposes). These files help us to measure the delay.

Before running the programs as the `TODO.md` file suggests, the VoiceSeeker configuration must be changed. The debug feature must be enabled and the delay property must be set up to 0. This is done by changing the `Config.ini` file as follows:

```
$ vi Config.ini
```

Set the following values for the `DebugEnable` and `RefSignalDelay` properties. The `DebugEnable` variable can be turned off when the calibration is done, but it is kept until the end for the sake of this document.

```
DebugEnable = 1
RefSignalDelay = 0
```

Save the file and quit Vi.

Launch `voice_ui_app` and AFE using the `VoiceSeeker` library, as described in the `TODO.md` file. It is important to have both the AFE and `voice_ui_app` running in the background. When one of them is not running, it can cause unexpected behaviors.

```
$ ./voice_ui_app &
```

To verify that the `voice_ui_app` is initialized properly, look at the log. If `voice_ui_app` prints the available VIT commands, the `voice_ui_app` initializes VoiceSpot and VIT properly.

```
$ ./afe libvoiceseekerlight &
```

To produce a known signal, use GST. Create a minimal pipeline using `gst-launch-1.0`. The command produces a periodic tick which makes it easier to measure the delay. Play the audio for a few ticks.

```
$ gst-launch-1.0 audiotestsrc wave=8 ! alsasink
```

The ticks are heard in the speakers.

For further information about any GST plugin, run the `gst-inspect-1.0` command.

```
$ gst-inspect-1.0 audiotestsrc #As an example
```

After playing the audio for a few ticks, stop the pipeline by pressing Ctrl + C and terminate AFE and `voice_ui_app`.

```
$ pkill afe
$ pkill voice_ui_app
```

The three mentioned audios should be generated and located in the `/tmp/` directory:

```
$ ls -lh /tmp/*.wav
-rw-r--r-- 1 root root 8.4M Mar 5 07:31 mic_in_delay_S0_E1.wav
-rw-r--r-- 1 root root 2.1M Mar 5 07:31 mic_out_S0_E1.wav
-rw-r--r-- 1 root root 4.2M Mar 5 07:31 ref_in_delay_S0_E1.wav
```

- The `mic_in_delay_S0_E1.wav` file contains the mixed signal (playback and voice) played from the speaker and recorded from the 8MIC board (delayed signal).
- The `mic_out_S0_E1.wav` file contains the clean voice (ideally null in this case, because no voice is recorded) obtained from AEC algorithm.
- The `ref_in_delay_S0_E1.wav` file contains the original playback stream (tick tone) without being recorded (signal with no delay).

Where:

- `_S0_ (Start minute)`: the minutes it started recording.
- `_E1_ (End minute)`: the minute it stops recording.

By default, the library records a pair of minutes (0-1, 2-3, 4-5, and so on). To change the default behavior, change the `MINUTE_INTERVAL_WAV_FILE` configuration macro (located in `<root_dir>/voiceseeker/src/SignalProcessor_VoiceSeekerLight.h`). For example, to record all the minutes on independent files, set up the macro with a value of 1.

Only the `mic_in_delay_S0_E1.wav` and `ref_in_delay_S0_E1.wav` files are needed to measure the delay. Copy those files to the host machine (using `scp` or any other method), open one of them with Audacity, and drag and drop the other wave file into the same Audacity window.

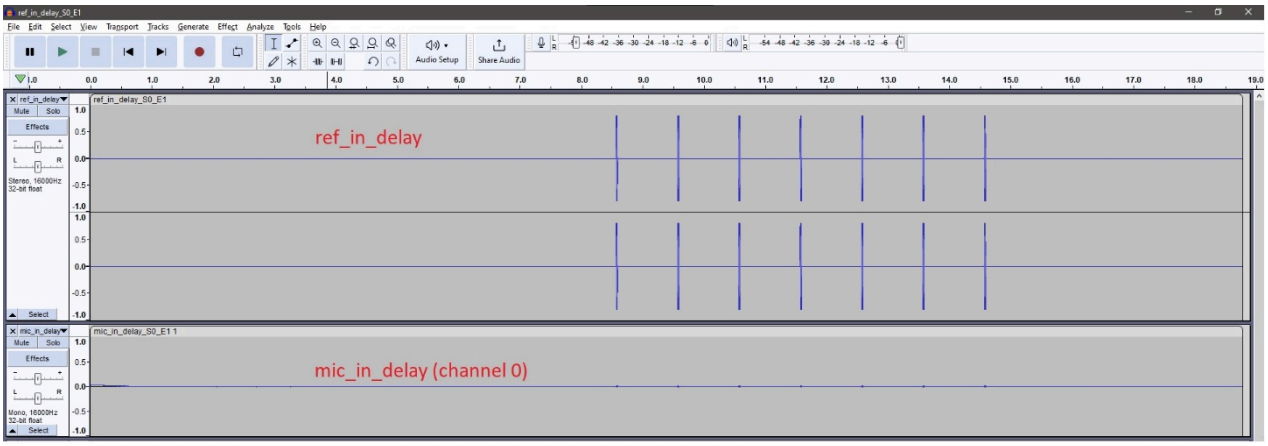

Figure 5. Audacity window

To achieve a better view of the data, perform the following steps to add some gain to the microphone input.

1. Select the four microphone channels.
2. Go to "Effect -> Volume and Compression -> Amplify".
3. A pop-up window appears. Check the “Allow clipping” box and set the amplification so that you clearly see the tick wave.

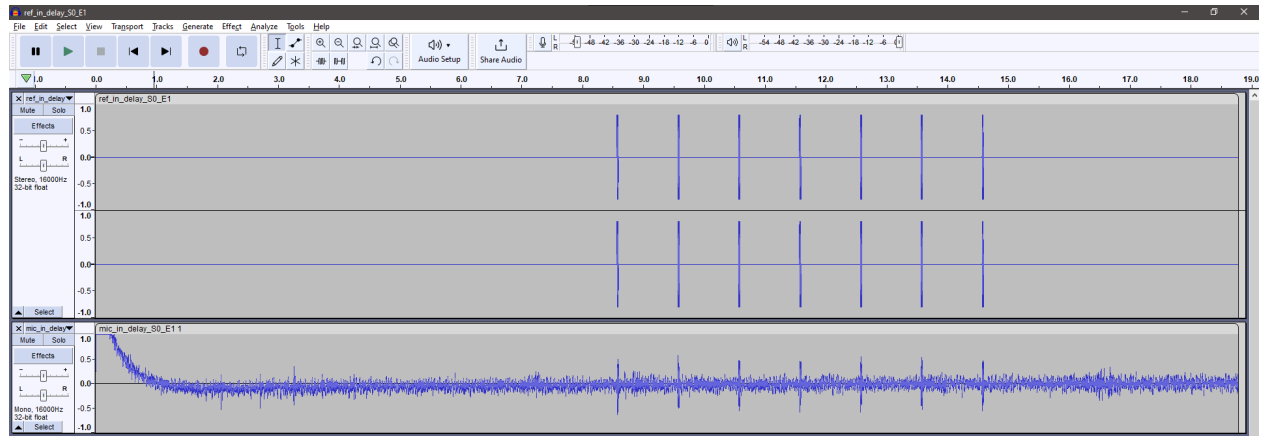

Figure 6. Tick wave

To achieve good accuracy, zoom into the beginning of the first tick on the reference signal and select all the samples between the reference point and its representation in the microphone channel.

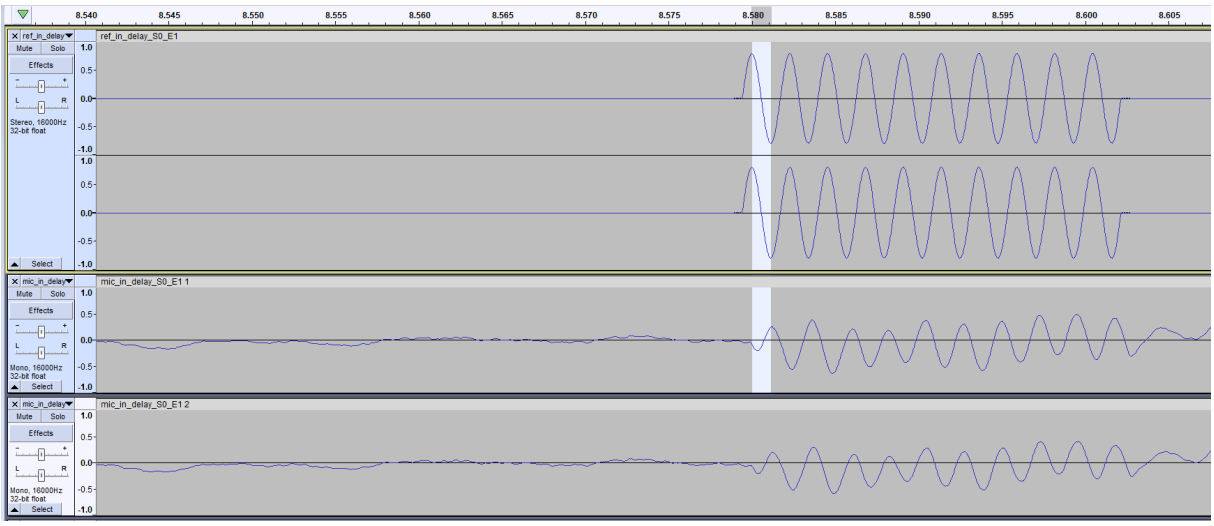

Figure 7. Microphone channel

The delay can be measured either in time or in sample units. VoiceSeeker works with the delay in samples. The audio editors usually allow time-frame measurements in sample units. To do so in Audacity, go to the bottom-left corner and change the option bar to “Start and Length of Selection”. Then you will see for how many samples the signal is delayed.

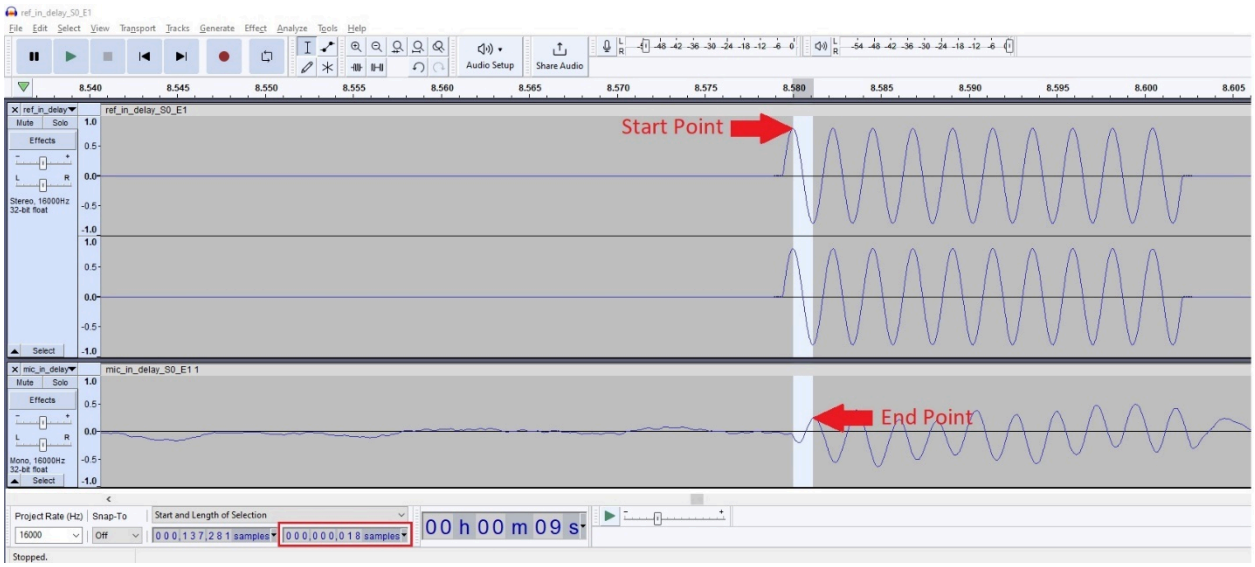

Figure 8. Signal delay

Set `RefSignalDelay` with this new value.

8.2 Beamforming

This section explains the performance of the beamforming capabilities of VoiceSeeker in conjunction with VoiceSpot.

Beamforming is the ability to highlight a section of a signal when it detects certain properties. VoiceSeeker uses beamforming when it detects a wake word.

The clearest way of seeing this feature is by playing a pure tone, trigger the wake word, and say a voice command from VIT.

Be sure to work with latest WAV files by removing the temporary files first.

```
$ rm -v /tmp/*.wav
```

Start VoiceSpot and VIT using `voice_ui_app` and run VoiceSeeker using AFE.

```
$ ./voice_ui_app &
$ ./afe libvoiceseekerlight &
```

Play a pure tone using GST.

```
$ gst-launch-1.0 audiotestsrc wave=0 ! alsasink
```

Say the wake word and voice command; for example, “Hey NXP! Mute”. Wait a few seconds and then repeat it. The wake word and the voice command can be changed.

Press "Ctrl + C" to stop GST and stop the processes, as you did previously.

```
$ pkill voice_ui_app
$ pkill afe
```

This time the required wave files are `mic_in_delay_S0_E1.wav` and `mic_out_S0_E1.wav`. In the first file, the voice sounds as if it was in the background and the second one sounds as if the voice moved to the front.

Copy those files to the host machine, open the `mic_out_S0_E1.wav` file and then drag and drop the other file.

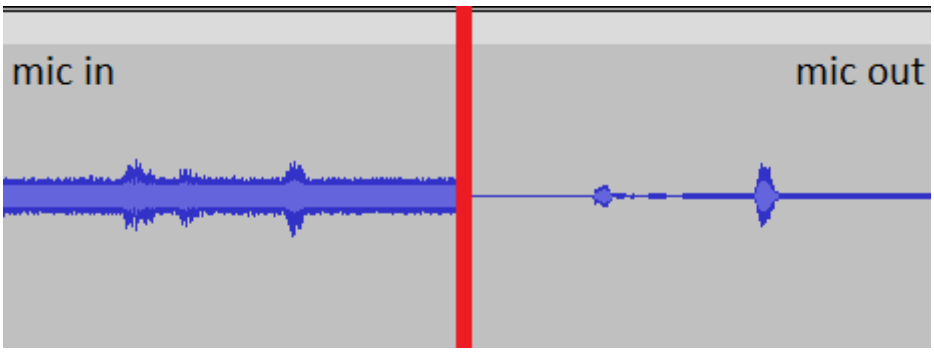

Figure 9. mic_in_delay_S0_E1.wav and mic_out_S0_E1.wav

Select all channels and apply some gain to them. Before playing the audio, you can see how the beamforming acts on the microphone output signal, isolating the voice when it detects it.

Another way to compare the audio is by playing each one in the solo mode by pressing the "solo" button on the left-hand side on each channel. When playing any channel from the microphone, the voice sounds as if it was in the background. It truly is, because the speakers are closer to the microphone. However, when playing the microphone output data, the voice seems to be closer to the microphones than the speakers.

8.3 External references

This section explains how the external signal feature can be tested using the current setup.

The idea is to create a virtual device that is in charge of writing to the speakers and feeding AFE with the references signal, as shown in Figure 10:

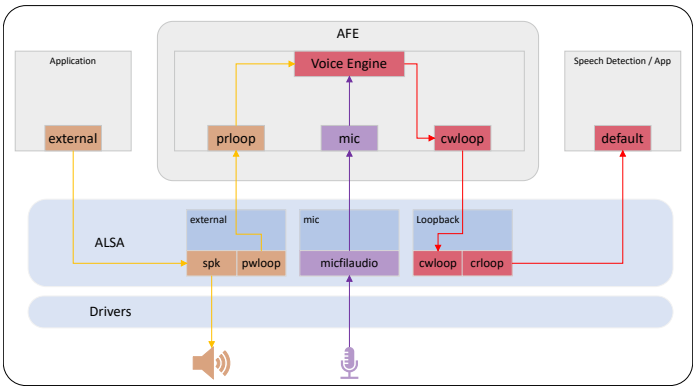

Figure 10. Virtual device

8.3.1 asound.conf

To test the feature, add the following devices to the `/etc/asound.conf` file. The idea is to duplicate the playback stream at the ALSA level, which is in charge of writing to the speakers and feeding AFE with the streamed data. The application must write to a different device than the default one, but it can be transparent to the application if the external device replaces the `plug:mix` device in the default pipeline.

Open the `asound.conf` file using an editor of your choice and add the following lines to it:

```
pcm.external {
        type plug
        slave.pcm "stereo2quad"
}

pcm.stereo2quad {
        type route
        slave.pcm "loopNspk"
        ttable.0.0 1
        ttable.1.1 1
        ttable.0.2 1
        ttable.1.3 1
}

pcm.loopNspk {
        type multi
        slaves.a.pcm "spk"
        slaves.a.channels 2
        slaves.b.pcm "pwloop"
        slaves.b.channels 2
        bindings.0 { slave a; channel 0; }
        bindings.1 { slave a; channel 1; }
        bindings.2 { slave b; channel 0; }
        bindings.3 { slave b; channel 1; }
}
```

Save the file and quit.

The above code creates three devices. The first device (`external`) is the first element of the chain and the one that the application must talk to. It is responsible for converting the format/rate to a supported output. The second device (`stereo2quad`) duplicates the channels. The third device (`loopNspk`) is where the main part is. This device is responsible for writing to two different sound cards. One is for the speaker and the other one opens the device that writes to the AFE input device.

The difference between using these devices and using the default one is that ALSA manages the playback thread. When running the following command, it causes the speakers to start playing even before AFE gets executed.

```
$ gst-launch-1.0 audiotestsrc wave=0 ! alsasink device=external
```

8.3.2 Running the pipeline

This section explains how to run the pipeline.

8.3.2.1 voice_UI

The `voice_ui_app` runs in the same way as earlier, it does not require any changes.

```
$ ./voice_ui_app &
```

8.3.2.2 AFE

Running AFE in the external mode is very easy, as described earlier on. The most important thing in this configuration is to know which device should be opened and AFE does the rest. In this case, `prloop` is the default device that must be opened to get the reference signal. However, running the program in the same way as before may produce an error, because the speaker device is already taken by another application. AFE must disable this device to avoid any errors at runtime. All of this is done by the following command:

```
$ ./afe libvoiceseekerlight prloop &
```

When AFE has opened the device that contains the reference data, execute the playback application:

```
$ gst-launch-1.0 audiotestsrc wave=0 ! alsasink device=external
```

Stop the application after a few seconds. Copy the `ref_in_delay_S0_E1.wav` file to the host machine and open it using Audacity to verify that the file is not empty. Also, executing `hexdump` is sufficient.

8.4 AEC enablement

This section describes how to build VoiceSeeker with the AEC enabled. A similar process is required to build VoiceSpot without timeout.

VoiceSeeker AEC provides an attenuation from 25 dB to 30 dB of the reference signal when it is properly configured and when there is no vibration in the system or any extra post processing in the signal that AFE is not aware of.

8.4.1 Getting the library

To enable the AEC (Acoustic Echo Cancellation), upgrade the VoiceSeeker library by contacting the NXP Voice Team at voice@nxp.com.

The Voice Team provide a folder for the requested platform, similar to those located in the `voiceseeker/platforms` folder in the imx-voiceui repository. After getting the folder, install it into the source code and compile the binaries again.

8.4.2 Installing the library

Clone the repository from GitHub:

```
$ git clone https://github.com/nxp-imx/imx-voiceui.git
```

Move to the latest branch:

```
$ cd imx-voiceui
$ git switch MM_04.09.00_2405_L6.6.y
```

Replace the `voiceseeker/platforms/iMX8M_CortexA53` folder with the new folder, which contains the full version of VoiceSeeker:

```
$ cp -r ../iMX8M_CortexA53/* ./voiceseeker/platforms/iMX8M_CortexA53
```

8.4.3 Compiling the library

The instructions to compile the library with the AEC enabled are in the readme.md file. However, setting up the AEC variable from the terminal is easier and requires fewer keypresses when building the image.

```
$ export AEC=1
$ make
```

After the previous step, the following message should be printed just before the compilation starts:

```
Building with AEC
```

Building the library generates the `release` folder, which contains the `voice_ui_app` binary and `libvoiceseekerlight.so.2.0`, along with other files that do not have to be deployed to the board, because they were not changed.

8.4.4 Board deployment

The only truly needed binary (in this case) is VoiceSeeker, but it is a good practice to overwrite both binaries.

```
$ scp release/libvoiceseekerlight.so.2.0 root@<BOARD_IP>:/usr/lib/nxp-afe/
$ scp release/voice_ui_app root@<BOARD_IP>:/unit_tests/nxp-afe/
```

When the binaries are installed on the target board, run them to verify that they work properly.

When VoiceSeeker is initialized, it dumps a configuration status, telling how it is configured. In that part, there is a property called `create_aec`, which has a value of "1" when AEC is enabled and a value of "0" when it is not.

8.5 Low power voice demo

Since this topic is too complex, it requires a dedicated document to cover it all. See *Low Power Voice UI Demo* (document AN13957) for how to enable it.

9 Appendix

9.1 Using NXP SWPDM with AFE

This section explains how to integrate the NXP SWPDM ALSA Plugin to the voice pipeline.

9.1.1 NXP SWPDM

NXP SWPDM is a solution for the i.MX 8MM and i.MX 8MN processors, which do not have a 32-bit width resolution required for voice-processing tasks. By adding this ALSA plugin to the pipeline, it increases the accuracy of the wake words and voice commands without significantly increasing the CPU load.

The SWPDM plugin uses a CIC filter algorithm to convert the PDM data from the microphones to a PCM format required for any audio postprocessing activities.

9.1.2 Adding SWPDM to the VoiceSeeker pipeline

This section explains the steps needed to integrate SWPDM to the VoiceSeeker pipeline. Although the 8M Plus does not require the plugin, it is compatible with it and the steps are the same regardless of the board (8M Plus, 8MM, or 8MN).

9.1.2.1 Using SWPDM

To use the SWPDM plugin, see the Community Post, which describes this topic and steps to integrate the plugin with AFE and `voice_ui_app`.

9.2 Microphone arrangement

This section explains how to edit the expected geometry of microphones to match the geometry of microphones located on the 8MIC-RPI-MX8 board. The same idea can be applied to fit any other microphone geometry.

9.2.1 Microphone routing

The 8MiC-RPI-MX8 board has eight microphones, numbered from zero to seven and placed as shown in Figure 11.

Figure 11. Microphones

The relative microphone positions are also needed to enhance the AEC. These positions are shown in Figure 12. In this case, the distances are measured from the center of the microphones in millimeters.

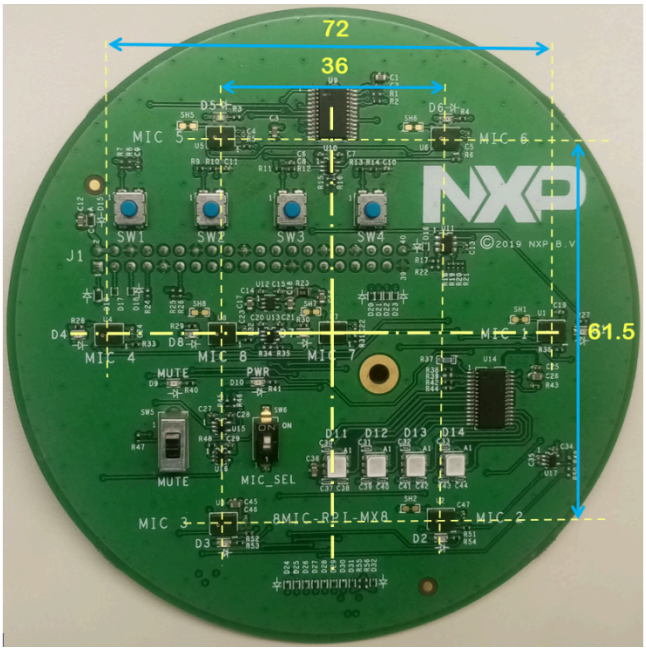

Figure 12. Microphone relative positions

Each microphone can be routed to any available channel of the recording audio stream by modifying the `/etc/asound.conf` file using `ttable`:

```
pcm.mic {
type route
slave.pcm "hw:micfilauio,0"
slave.channels 8
ttable.0.0 1
```

```
ttable.1.1 1
ttable.2.2 1
ttable.3.3 1
ttable.4.4 1
ttable.5.5 1
ttable.6.6 1
ttable.7.7 1 }
```

The `ttable` property can reroute a physical microphone to any audio stream channel as follows:

The legend for the above snippet is `ttable.A.B 1`, where:

- `A` represents the channel index used in the audio stream.
- `B` represents the physical microphone index on the board.
- `1` represents the number of microphones.

For example, line `ttable.3.6 1` routes the audio signal coming from microphone 6 to channel 3.

9.2.2 Microphone position configuration

AFE uses the `config.ini` file to set up the microphone relative coordinates.

The `config.ini` file must be also edited. The following code snippet shows a typical microphone coordinate setup:

```
mic0 = 0.00, 0.00, 0.00
mic1 = 36.00, 0.00, 0.00
mic2 = -18.00, 30.75, 0.00
mic3= -18.00, -30.75, 0.00
mic4 = 18.00, -30.75, 0.00
mic5 = -36.00, 0.00, 0.00
mic6 = 18.00, 30.75, 0.00
mic7 = -18.00, 0.00, 0.00
```

Number `x` in the `micx` variable refers to the virtual microphone channel in the audio stream and not the physical microphone index. If you want to set up the coordinates of physical microphone three, route it to a channel (using channel five as an example) in the `asound.conf` file:

```
ttable.5.3 1
```

Set the coordinates to the same channel (channel five) in the `config.ini` file:

```
mic5 = -35.0, 18.0, 0.0
```

The coordinates are written as follows:

```
micX = x-coordinate, y-coordinate, z-coordinate
```

The coordinates are measured in millimeters and they are relative to an arbitrary point. The location of this point can be anywhere, in a microphone, in a point on the board, or even in a location outside the board.

For simplicity reasons, a microphone can be chosen as the origin by setting its coordinates to 0, 0, 0 and writing the coordinates of the other microphones in relation to the origin.

9.2.3 Microphone arrangement example

This example shows the configuration to use a custom set of microphones (the ones circled in purple). The intention of this example is to route microphones 6, 5, 3, and 1 to channels, 0, 1, 2, and 3, respectively.

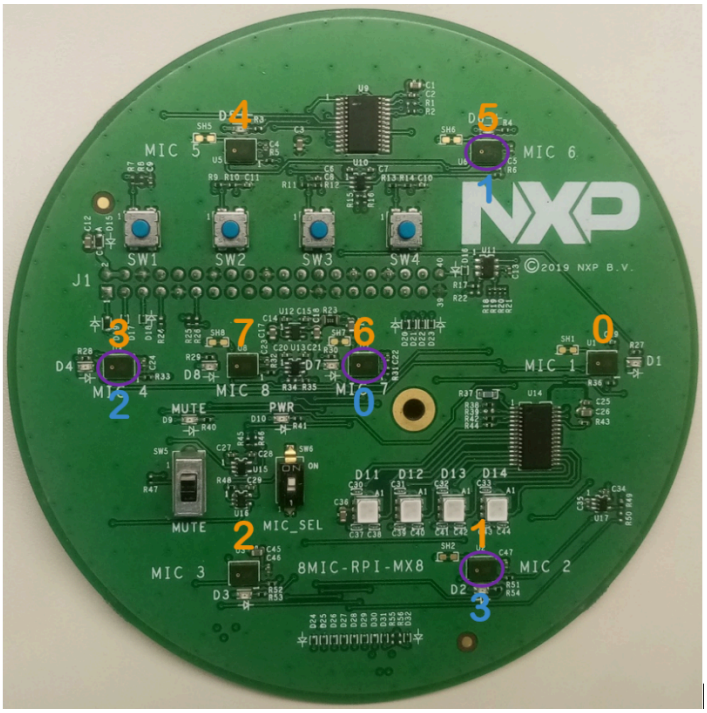

Figure 13. Microphone numbers in yellow and desired channels in blue

To route the microphones to the desired channels, use the following `ttable` configuration in the `asound.conf` file:

```
pcm.mic {
type route
 slave.pcm ”hw:micfilaudio,0”
 slave.channels 8
 ttable.0.6 1
 ttable.1.5 1
 ttable.2.3 1
 ttable.3.1 1
}
```

To set the microphone corresponding coordinates, use the following configuration in the `config.ini` file:

```
mic0 = 0.0, 0.0, 0.0
mic1 = 18.0, 30.0, 0.0
mic2 = -35.0, 0.0, 0.0
mic3 = 18.0, -30.0, 0.0
```

In this case, the physical microphone six on channel zero is chosen as the origin, and the other three channel positions are measured using this microphone as the origin. Figure 14 shows the four chosen microphones and their relative coordinates.

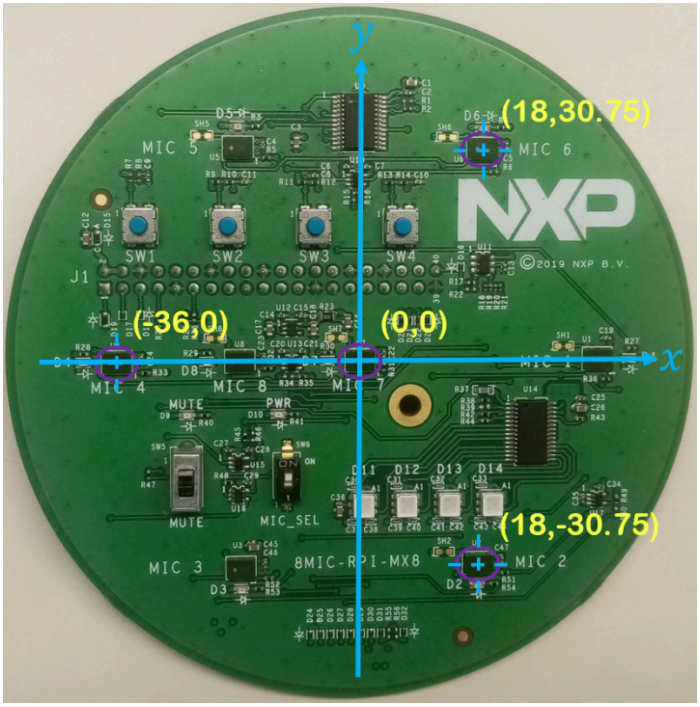

Figure 14. Four chosen microphones and their relative coordinates

10 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11 Revision history

Table 8 summarizes the revisions to this document.

Table 8. Revision history

Document ID	Release date	Description
AN14276 v.1.0	24 June 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

i.MX — is a trademark of NXP B.V.

MATLAB — is a registered trademark of The MathWorks, Inc.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Contents

1 Introduction 2
2 Purpose 2
3 Overview 2
4 Materials 3
4.1 Software requirements 3
4.2 Hardware requirements 3
5 Voice-UI framework 3
5.1 Integration to NXP Linux BSP 3
5.1.1 Architecture 4
5.1.1.1 External references signal 5
5.1.2 NXP AFE 5
5.1.2.1 asound.conf 6
5.1.3 VoiceSeeker 7
5.1.3.1 Config.ini 9
5.1.4 VoiceSpot 10
5.1.4.1 Model 10
6 Installation guide 10
6.1 Hardware checklist 10
6.1.1 For i.MX 8M Plus 10
6.2 Setup view 11
6.3 Software installation 12
6.3.1 Software checklist 13
7 Software setup 13
7.1 Selecting the device tree for the 8MIC-RPI-MX8 board 13
7.2 Installing missing drivers 14
7.3 Editing the ALSA asound configuration for VoiceSeeker 14
8 Audio lab guides 14
8.1 Delay measurement 14
8.2 Beamforming 17
8.3 External references 18
8.3.1 asound.conf 19
8.3.2 Running the pipeline 20
8.3.2.1 voice_UI 20
8.3.2.2 AFE 20
8.4 AEC enablement 20
8.4.1 Getting the library 20
8.4.2 Installing the library 21
8.4.3 Compiling the library 21
8.4.4 Board deployment 21
8.5 Low power voice demo 21
9 Appendix 22
9.1 Using NXP SWPDM with AFE 22
9.1.1 NXP SWPDM 22
9.1.2 Adding SWPDM to the VoiceSeeker pipeline 22
9.1.2.1 Using SWPDM 22
9.2 Microphone arrangement 22
9.2.1 Microphone routing 22
9.2.2 Microphone position configuration 24
9.2.3 Microphone arrangement example 25
10 Note about the source code in the document 26
11 Revision history 26
Legal information 28