AUD-ESP-00518

IA8201 Voice Wake Solution Software Guide

This document provides the following key software details from SW perspective, to enable customers to use IA8201 to realize Voice Wake solution in their products:

Table of Contents

Chapter 1: Introduction

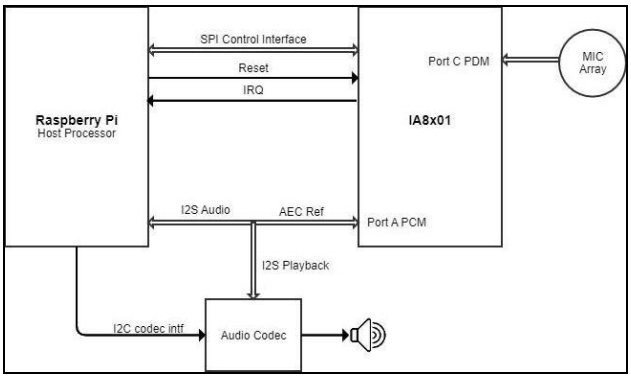
This document provides the following software details, to enable customers to use IA8201 to realize Voice Wake solution in their products:

1. Solution architecture of the Reference IA8201 Voice Wake Solution running on the EVM Kit. The reference solution realizes 2 use cases:
    - AVS integration with IA8201 providing "Always ON" Voice Wake feature
    - A simple Voice Wake App with IA8201 providing "Always ON" Voice Wake feature, which can be reference for any custom Voice Interface solutions
2. Steps to follow to integrate the "Reference IA8201 Voice Wake Solution" into a customer product
3. Supported customizations that can be done on top of the "Reference IA8201 Voice Wake Solution"

Chapter 2: Solution Architecture

This chapter provides instructions on how to integrate IA8201 with the QCC5124 platform.

2.1 System Architecture



2.2 Software Architecture

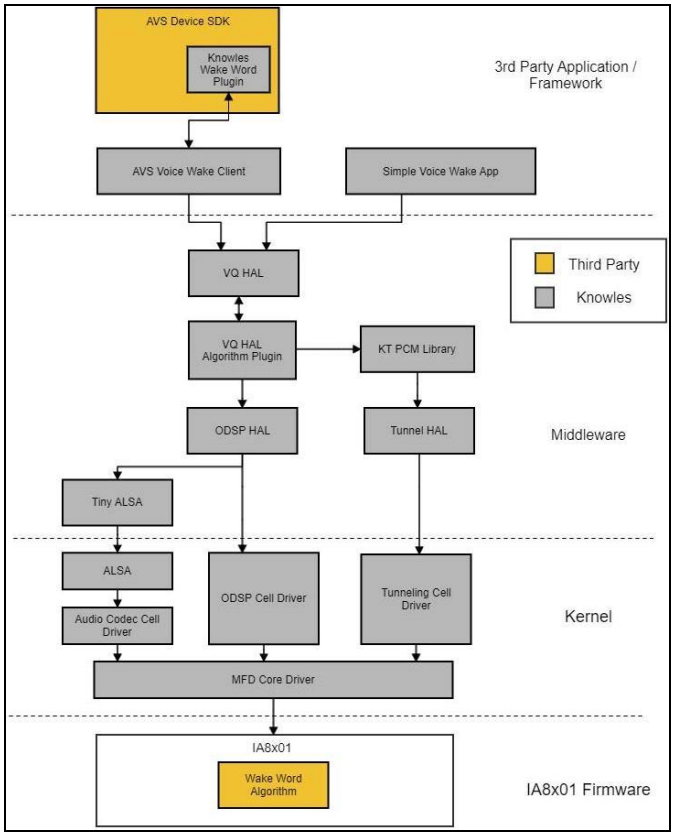


Figure 1

2.3 Software Components

2.3.1 IA8x01 Firmware

The IA8x01 Firmware provides the infrastructure and resources for the algorithm plugins to run on the DSP core

2.3.2 Wake Word Algorithm.

The wake-word algorithm monitors the audio stream from the Microphones looking for the wake-word

2.3.3 MFD Core Driver

The IA8x01 kernel driver follows Multi-Function Device (MFD) model. Kernel component is divided into multiple cell device drivers: Codec driver, Tunnel driver, ODSP (Open DSP) driver. Core cell driver implements the management and control of the chip and the client drivers would make use of this through exposed APIs. The individual cell drivers implement their respective functionalities and expose them to user space through their device nodes.

2.3.4 ODPS Cell Driver

The Open DSP driver exposes a device node and IOCTLs for the HAL code to communicate with the Algorithm components running on the DPS chip. It exposes different APIs to download, create and execute different algorithm blocks on the DSP chip abstracting all lower-layer operations.

2.3.5 Tunnel Cell Driver

The Tunnel cell driver exposes the APIs to transport audio/non-audio data over the underlying control interface between the IA8x01 DSP chip and the Host Processor. This driver exports a device node for the HAL to access the tunnel data stream from the DSP. The tunnel driver supports multiple endpoints and multiple clients to avail the data from the chip concurrently.

2.3.6 Audio Codec Cell Driver

The codec audio cell driver implements the audio related functionalities of of Knowles's DSP. The codec driver is an ALSA-compliant driver and exposes Audio controls (i.e Kcontrols) to setup audio routes on the DSP for various use cases.

2.3.7 VQ HAL

VoiceQ HAL or VQ HAL layer implements the APIs that are required for Voice Recognition. This layer provides some simple API's to start/stop the keyword detection and returns an event upon keyword detection. It also provides API's to tunneling out the audio data after keyword detection.

2.3.8 ODSP HAL

This HAL layer interacts with the IA8x01 kernel ODSP Cell driver and provides simple user space API's that can be used by its clients to interact with the kernel driver. IA8x01is built as an Open DSP platform. Using the SDK provided for IA8x01, one can package their algorithms into plugins. These plugins can be loaded, created and run one at a time or together in pipeline or in parallel. The ODSP HAL provides API's to load, create, destroy and unload plugins on the IA8x01 platform. It also provides certain utility API's that can provide some additional information that the plugins might need.

2.3.9 Tunneling HAL

Tunneling is a mechanism for transporting the data from IA8x01 to the host device via the control bus. Tunneling can transport audio data or any type of opaque data. This HAL layer provides APIs that the application can use to get the data from the device.

2.3.10 KT PCM Library

This library provides ALSA like API's that can be used by the application to retrieve audio data from Chelsea. This library uses the Tunneling HAL API's and tunnels out the required audio data and strips out the tunnel headers before providing the raw audio data to the application.

2.3.11 HAL Algorithm Plugin

The APIs required to support an underlying voice wake algorithm are written as plugin files. This enables us to use different voice wake algorithms using the same HAL libraries.

2.3.12 Simple Voice Wake Application

Knowles' Simple Voice Wake application is a sample application that shows the use of the VQ HAL. It will load the VQ HAL library and run the low power VQ use case. On keyword detection, it shall stream the audio data for 5 seconds (as an example). Once the audio streaming is completed, it then goes back to keyword detection mode.

2.3.13 AVS Device SDK, Knowles Wake Word Plugin and AVS Voice Wake Client

The AVS Device SDK component has been taken from the official amazon GitHub project. We have added a wake word plugin in to the SDK code to add support for the Knowles DSP wake word detection. The Wake word event are communicated over socket communication to the wake word plugin running in the SDK.

The AVS Voice Wake Client interacts with the VQ HAL to setup the voice recognition route and it also connects with the Knowles Wake Word plugin, sending keyword detection events and the audio data that is streamed after the detection of the keyword to the Knowles Wake Word plugin

2.4 Use Case Execution Sequences

2.4.1 AVS

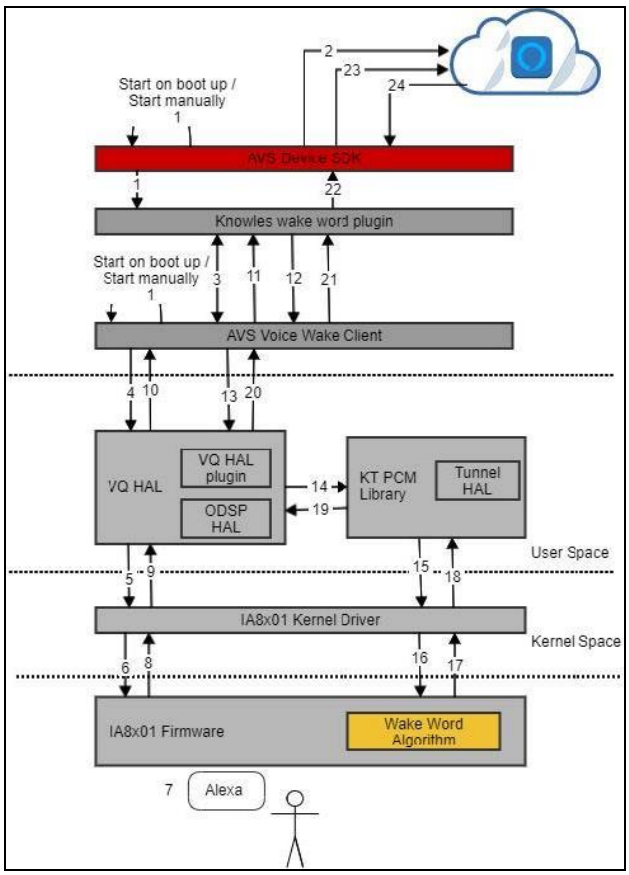


Figure 2

1. or load The AVS VoiceWake application and the AVS Device SDK sample application can be started manually or loaded at bootime. As a part of the AVS Device SDK setup, it enables the Knowles Wake Word Plugin.
2. The AVS Device SDK, requests the user to register the current device with the AVS cloud services.
    a. The AVS VoiceWake application and the Knowles Wake-Word plugin setup bi-directional socket communication. AVS VoiceWake application
    b. Initializes the VQ HAL and starts the keyword recognition
    c. The VQ HAL talks to the kernel driver to start the keyword recognition.
    d. The kernel driver sends the necessary commands and the data to start the recognition on the Chlesea hardware
3. User utters the keyword
    a. IA8x01 firmware raises an interrupt
    b. The IA8x01 kernel driver catches intercepts the interrupt raised by the firmware and sends keyword detection event to the VQ HAL
    c. The VQ HAL informs the AVS Voice Wake application about the keyword detection via a callback
    d. The AVS Voice Wake application sends the keyword detection event message to the Knowles Wake Word plugin.
    e. The Knowles Wake Word plugin requests the AVS Voice Wake application to start sending the audio data associated with the keyword.
    f. AVS Voice wake requests the VQ HAL to provide the audio data.

g. The VQ HAL invokes the KT PCM library to start tunneling out the data
h. The KT PCM library in turn talks to the Kernel driver to get the audio data
i. The Kernel driver sends the streaming command to the firmware.
j. The Firmware sends out the audio data via a tunnel.
k. The Kernel driver reads this tunnel data and provides the data to the KT PCM library
l. The KT PCM library parses the tunnel data to get the raw audio data and sends the raw audio data to the VQ HAL
m. The VQ HAL passes the raw audio data to the AVS Voice Wake application.
n. The AVS Voice Wake application sends this raw audio data to the Knowles Wake Word Plugin via the socket.
o. Knowles Wake Word plugin writes this raw audio data into the circular buffer maintained by the AVS Device SDK and informs it of the keyword detection.
p. The AVS Device SDK sends the raw audio data to the AVS cloud service.
q. The AVS Cloud service analyses the data and informs the AVS Device SDK whether it was a valid keyword interrupt or an invalid. If it was a valid keyword interrupt then it also sends a response related to the query done by the user in step 7. AVS Device SDK sample application takes this response is and it either prints it to the console or plays it out via the speaker.

2.4.2 Simple Voice Wake App

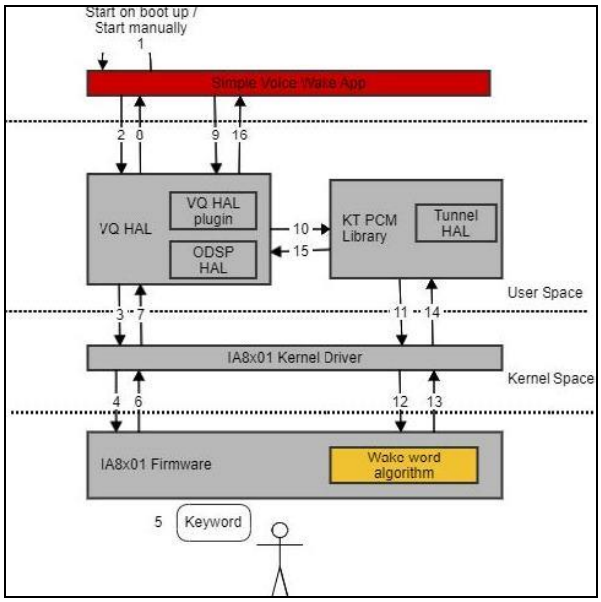


Figure 3

1. The VoiceWake application can be started manually or loaded at bootime
    a. The Voice Wake application initializes the VQ HAL and starts the keyword recognition.
    b. The VQ HAL communicates with the kernel driver to start the keyword recognition.
    c. The kernel driver sends the necessary commands and data to the Chelsea hardware to start the keyword recognition.
2. Utter the keyword.
    a. IA8x01 firmware raises an interrupt.
    b. The IA8x01 kernel driver intercepts the firmware's interrupt and sends the keyword detection event to the VQ HAL.
    c. The VQ HAL informs the Voice Wake application about the keyword detection through a callback.

d. The Voice Wake application can start receiving the audio data after the keyword was uttered by requesting for the same from the VQ HAL
e. The VQ HAL invokes the KT PCM library to start tunneling out the data
f. The KT PCM library in turn talks to the Kernel driver to get the audio data
g. The Kernel driver sends the streaming command to the firmware.
h. The Firmware sends out the audio data via a tunnel.
i. The Kernel driver reads this tunnel data and provides the data to the KT PCM library
j. The KT PCM library parses the tunnel data to get the raw audio data and sends the raw audio data to the VQ HAL
k. The VQ HAL passes the raw audio data to the Voice Wake application.

2.4.3 Integration Points

The "Reference IA8201 Voice Wake Solution" has a typical layered SW architecture spanning Application, Middleware & Kernel layers.

Customers can use this layered SW stack and use majority of the SW components provided in the reference solution as-is. So the recommended integration point to integrate this SW stack into a customer product SW stack is the "VQ HAL" API.

"VQ HAL" API abstracts the Host interaction with IA8201 DSP and provides a simple & easy to use API to integrate into customer product stack. The "VQ HAL" API provides all necessary functions to control the Voice Wake feature supported by IA8201

2.4.4 VQ HAL API

vq_hal_init
```
struct vq_hal* vq_hal_init(bool enable_host_side_buffering, struct vq_hal_config vqhc);
```

Description: This API initializes the VQ HAL library.

Arguments

`enable_host_side_buffering`: This enables the VQ HAL to buffer on the host side. On keyword detection, the VQ HAL starts buffering automatically and when the application starts to read the audio data it gives out the audio data from the internal buffer that is held by the VQ HAL first. This helps in scenarios where the application takes a long time to start the audio streaming after the keyword is detected, which could lead to audio data loss.

`vq_hal_config`: VQ HAL configuration structure. Please see Data Structures and Callback Functions section for more details.

Return value: On success, returns handle to the vq_hal structure , and on failure returns NULL.

vq_hal_deinit
```
int vq_hal_deinit(struct vq_hal *hdl);
```

Description: This API de-initializes the VQ HAL library.

Arguments

`hdl`: Pointer to a valid vq_hal structure.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_start_recognition
```
int vq_start_recognition(struct vq_hal *hdl, vq_hal_music_status status);
```

Description: This API downloads the VQ algo package, and any additional packages required for the use case to the DSP that are required for the usecase and sets up the route and startsthe keyword recognition.

Arguments
`hdl` – Pointer to a valid vq_hal structure.

Status – Music is On/Off.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_pause_recognition	`int vq_pause_recognition(struct vq_hal *vq_hdl);`

Description: This API will pause the voice recognition. The microphone will not be used at this point.

Arguments
`hdl` – Pointer to a valid vq_hal structure.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_resume_recognition	`int vq_resume_recognition(struct vq_hal *vq_hdl);`

Description: This API will resume the voice recognition. This API can be called only after the vq_pause_recognition has been called. Calling this API will allow it to use the microphone and perform the voice recognition.

Arguments
`hdl` – Pointer to a valid vq_hal structure.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_stop_recongition	`int vq_stop_recognition(struct vq_hal *vq_hdl);`

Description: This API stops the keyword recognition and it destroys all the VQ related plugins and packages and tears down the route.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_register_cb	`int vq_register_cb(struct vq_hal *vq_hdl, void *cookie, vq_hal_event_cb cb);`

Description: This API registers a callback for the application. The application can receive Keyword detection, Firmware crash and recovery events and error events via this callback.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

`Cookie` – An opaque data object that will be returned back to the application in the callback function.

`cb` – Pointer to the callback function

Return value: On success returns 0 and on failure returns a value less than 0.

vq_unregister_cb	`int vq_unregister_cb(struct vq_hal *vq_hdl);`

Description: This API unregisters the callback from the application. After this API is called, no events will be sent to the application from the VQ HAL library.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_start_audio	`int vq_start_audio(struct vq_hal *vq_hdl, bool is_multiturn);`

Description: This API asks the VQ HAL to start the audio data streaming. This API can be called just after the keyword has been detected or before the keyword is detected. If the API is called before keyword is detected then flag is_multiturn should be set.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

`is_multiturn` – This flag should be set if the audio needs to be started before the keyword is detected. Depending on the underlying route, the audio data will be raw data form the mic or the processed output from the Front End processing algo module based on the algo configuration. If the API is called after the keyword is detected then this flag should not be set.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_stop_audio	`int vq_stop_audio(struct vq_hal *vq_hdl);`

Description: This API stops the audio streaming that is in progress.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

Return value: On success returns 0 and on failure returns a value less than 0.

vq_read_audio	`int vq_read_audio(struct vq_hal *vq_hdl, void *buf, int buf_size);`

Description: This API reads the audio data from the VQ HAL.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

`buf` – Emtpy buffer into which the audio data will be copied into.

`buf_size` – Size of the empty buffer that is available for the VQ HAL to copy the audio data into.

Return value: Number of bytes read into the buffer.

vq_get_audio_frame_length	`int vq_get_audio_frame_length(struct vq_hal *vq_hdl);`

Description: This API returns the audio frame length in milliseconds.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

Return value: Audio frame length.

vq_set_music_status	`int vq_set_music_status(struct vq_hal *vq_hdl,`

```
vq_hal_music_status status);
```

API Name: vq_set_music_status

Description: This API informs the VQ HAL when the music playback starts and stops. The VQ HAL will use this information to update its routes based on this information.

Arguments
`vq_hdl` – Pointer to a valid vq_hal struture.

`Status` – Depicts the current music playback status.

Return value: On success returns 0 and on failure returns a value less than 0.
Data structures and callback function

Description:

```
struct vq_hal;
typedef enum {
EVENT_KEYWORD_RECOGNITION = 0,
EVENT_FW_CRASH,
EVENT_FW_RECOVERED,
EVENT_ERROR
} vq_hal_event_type;
typedef enum {
MUSIC_PLAYBACK_STARTED,
MUSIC_PLAYBACK_STOPPED
} vq_hal_music_status;
typedef enum {
ONE_MIC,
TWO_MIC,
THREE_MIC,
FOUR_MIC
} vq_hal_mic_configurations;
typedef enum {
NORMAL_MODE, // No power optimization done
OPTIMAL_MODE // Optimized for power when listening for keywords
} vq_hal_power_mode;
struct vq_hal_config {
vq_hal_mic_configurations mc; // Mic configuration for the VQ HAL route
vq_hal_power_mode pm; // Power mode to run in
int dev_id; // Device id for the usecase
};
struct event_data {
int kw_id; // Keyword id
int start_frame; // Start frame number where the keyword starts
int end_frame; // End frame number where the keyword ends
int interrupt_frame; // Frame number when detection event/interrupt is
generated
float confidence_lvl; // Confidence level of the detected keyword
};
```

- `vq_hal` – The VQ HAL private structure. On successful init, a pointer to this structure is returned to the application. The application should pass the same handle when using the VQ HAL API's.

- Power Modes -
    - NORMAL_MODE – No power optimization done
    - OPTIMAL_MODE - Optimized for power when listening for keywords

Mic configurations - Specifies the number of microphones to be used for a given wake word algorithm. Note - Not all wake word algorithms provide the option to change the microphone cofigurations.

- VQ HAL configuration - Provide more information on microphone configurations to be used and power mode. The device id that needs to be passed here is 0.
- Event types –
    - EVENT_KEYWORD_RECOGNITION – This event is sent after keyword detection.
    - EVENT_FW_CRASH – This event is sent if a firmware crash has been detected.
    - EVENT_FW_RECOVERED – This event is sent after the firmware is recovered from a firmware crash.
    - EVENT_ERROR – This is a generic event is sent when an VQ HAL runs into an error.
- Music status -
    - MUSIC_PLAYBACK_STARTED – Indicates that the music playback is currently in progress or just started.
    - MUSIC_PLAYBACK_STOPPED – Indicates that the music playback is currently not in progress or just stopped.
- event_data – This is data structure of the event that would be returned in the callback function on successful keyword detection. The description of the fields are as below –
    - kw_id – ID of keyword that was detected. Currently not useful as there is only one keyword that is loaded.
    - start_frame – Indicates the frame number from which the keyword starts.
    - end_frame – Indicates the frame number at which the keyword ends
    - interrupt_frame – Indicates the frame number when detection event is generated.
    - confidence level – Indicates the confidence level of the keyword detected

```
/**
* Prototype of the VQ Callback function
*
* Input - cookie - Opaque pointer that was passed during the callback
* registration
* - event - Type of event
* - event_data - Event data associated with event
* Output - 0 on success, on failure < 0
*/
```

```
typedef int (*vq_hal_event_cb)(void *cookie,
vq_hal_event_type event,

void *event_data);
```

- `vq_hal_event_cb` – Prototype of the callback function.
    - `cookie` – Pointer to an opaque data which was passed during the callback function registration.
    - `event` – Type of the event, currently there are 4 events that can be sent –
    - `event_data` – Additional event data.

Chapter 3: Integration Process

3.1 Overview

The "Reference IA8201 Voice Wake Solution" has several SW components that can be used as-in in the customer product when integrating IA8201.

However few components in the "Reference IA8201 Voice Wake Solution" are specific to the EVM HW. So these components can be used as a reference and appropriately modified according to the end product HW.

This section provides details on what is available in the SW package and how to integrate the SW components from this package into a target product SW.

RaspberryPi is the Host processor used in the EVM HW on which the "Reference IA8201 Voice Wake Solution" is running. So we have used Raspberry Pi to illustrate HW specific aspects in the SW components where needed. This is especially relevant for the Kernel/Driver.

3.2 SW Package

The Knowles software package contains all the source code packages, firmware and algorithm binary files

3.2.1 Software Package Directory structure

```
source/
├── chelsea-iot-middleware                  # Knowles HAL Libraries and Console Apps
│   └── source
│       ├── external
│       │   ├── avs-device-sdk              # AVS Device SDK with Knowles Wake word
support
│       │   └── tinyalsa                    # External tinyalsa lib
│       │
│       └── knowles
│           ├── console_apps                # Knowles Reference Console Apps
│           │
│           ├── libs                        # Knowles VQ HAL
│           │
│           └── voice_framework             # Knowles ODSP and Tunnel HAL
│
├── vq_hal_algorithm_plugins                # VQ HAL Algorithm plugins (Amazon, Retune
and Sensory Algo)
│
├── firmware_binaries                       # Knowles Firmware and Algorithm binaries
wakeword_algorithm_plugins
│   └── audience
│       └── ia8x01
└── kernel
```

```
    ├── adnc                                # Knowles ADNC Host Driver
    │   ├── drivers
    │   │   ├── mfd
    │   │   └── misc
    │   │
    │   └── sound
    │       └── soc
    │           └── codecs
    │               └── adnc                # Knowles ALSA CODEC driver
    └── dts                                 # Knowles Device Tree files
```

3.2.2 Kernel space integration

The Knowles software package contains the host software driver code as a multifunction device driver to support the IA8x01 DSP under the Linux Driver framework.

3.2.3 Kernel Driver Source code Organization

Knowles' Kernel Driver must be installed in linux kernel codebase in multiple places. The source encompasses following four directories:

drivers/mfd/adnc	Contains driver source for core, tunnel, and debug cell drivers.
drivers/misc/adnc	Contains driver entry point code for odsp and module cell drivers.
include/linux/mfd/adnc	Contains common headers shared between all cell drivers.
sound/soc/codecs/adnc	Contains source for codec cell driver.

3.2.4 Steps to integrate Kernel Driver

The Knowles ADNC multifunctional driver can be statically compiled into the Linux kernel image or built as a Loadable Kernel Module (LKM).

3.2.4.1 ADNC driver source structure

drivers/mfd/adnc/	Contains the core MFD driver files.
drivers/misc/adnc/	Contains the ODSP cell driver and tunnel cell driver.
sound/soc/codecs/adnc/	Contains the ALSA audio codec driver.

3.2.4.2 Makefile and another supporting file for LKM

We have added a standard LKM top-level makfile to support out of tree kernel module build. This also needs two additional files to define and pass all the defconfig macros as there don't have the Kbuild support during the out-of-the-tree builds.

- Makefile
- config.mk
- out-of-tree-autoconf.h

3.2.4.3 Defconfig changes for Knowles Driver

For Knowles' kernel driver to compile statically, the following changes are needed in defconfig file of the for host platform to compile Linux kernel. These macros have to be updated in the config.mk and autoconf.h file instead of the host defeconfig files for the out-of-the-tree build.

defconfig

```
# generic macros for IA8x01
CONFIG_MFD_IAXXX=y
CONFIG_MFD_IA8X01=y
CONFIG_SND_SOC_IAXXX=y
CONFIG_MFD_IAXXX_TUNNEL_POLL=y

#SPI bus specific
CONFIG_MFD_IAXXX_SPI=y

#I2C bus specific
CONFIG_MFD_IAXXX_I2C=y

#UART bus pecific
CONFIG_MFD_IAXXX_UART=y
CONFIG_IAXXX_UART_HW_FLOW_CONTROL=y
```

Please note that the out-of-the-tree LKM build need the defconfig and other macros to be specified in the config.mk and out-of-tree-autoconf.h

3.2.4.4 Makefile Changes for out of tree LKM build

The makefile variables have to be updated with the host systems cross compiler toolchain patch and the kernel source/header details for the build system to successfully build the modules.

```
# Specify the location of the Compiler (or Cross compiler toolchain)
CROSS:=${CURR_PWD}/../tools/gcc-linaro-7.5.0-2019.12-x86_64_arm-linux-
gnueabihf/bin/arm-linux-gnueabihf-

# specify the location of the Kernel Build Files (kernel headers, configuration
files, and needed object files)
KERNEL_DIR=${CURR_PWD}/../linux

# specify the location where the Ia8x01 Kernel modules need to be copied
MODULE_OUT=${CURR_PWD}/../deploy/images/raspberry-pi/ia8x01-ko_modules/
```

3.2.4.5 Config.mk & out-of-tree-autoconf.h

These are the two special files used for out-of-the-tree LKM builds. As we don't have the Kbuild support from the kernel build system, we need to pass all the defconfig macros to both the child makefiles and source files in this way.

Any changes in configuration need to be propagated to these file got generating a proper IA8x01 LKM modules.

config.mk

```
# Config file to specify Ia8x01 Driver config parameters
# retain CONFIG_SELECT at the beginning of the line

# select config to build the driver as kernel module
CONFIG_SELECT:=CONFIG_MFD_IAXXX=m

# select spi driver
CONFIG_SELECT+=CONFIG_MFD_IAXXX_SPI=y

# select config to build codec driver as kernel module
CONFIG_SELECT+=CONFIG_SND_SOC_IAXXX=m

# select config option to build ADAU1772 adapter (Honkers codec) as kernel module
CONFIG_SELECT+=CONFIG_IAXXX_SND_SOC_ADAU1772=m

[...]
```

These macros have to be passed to the source manually using a header to overcome no Kbuild support in case of out-of-the-tree build.

autoconf.h

```
/* like include/generated/autoconf.h from the kernel tree, comment out what  you
don't want (setting it to zero wont work) */

#define CONFIG_MFD_IAXXX_SPI                    1
#define CONFIG_SND_SOC_IAXXX                    1
#define CONFIG_IAXXX_SND_SOC_ADAU1772           1
#define CONFIG_MFD_IAXXX_DISABLE_RUNTIME_PM     1
```

3.2.4.6 Knowles ADNC Driver Loadable Kernel Modules (LKM)

After a successful build, the final kernel module (.ko) will be generated in the out directory mentioned in the makefile. Copy these modules into the target platform at desired locations.

Note that the kernel modules must be inserted (insmod/modprobe) in order shown above.

```
iaxxx-mfd-core.ko             # MFD Core
iaxxx-tunnel-driver.ko        # ODSP Cell driver
iaxxx-odsp-driver.ko          # Tunnel Cell driver
iaxxx-codec-driver.ko         # ASoC Codec Driver
iaxxx-alsa-tunnel-driver.ko   #(Optional module for alsa pcm tunnel)
```

3.2.5 Firmware and Algorithm Binaries

The Knowles ADNC driver will be expecting the IA8x01 Firmware and algorithm packages in the firmware folder to be preset. The firmware binary will be auto downloaded by the Host driver during the bus driver probe and the algorithm packages will be downloaded during the audio route setup initiated by the VQ HAL.

```
└── audience
    └── ia8x01
        ├── AmazonWWCreateConfig.bin                                  #
Amazon Wake word algo configuration file
        ├── AmazonWWPackage.bin                                       #
Amazon Wake word algo package
        ├── WR_250k.en-US.alexa.bin                                   #
Amazon Alexa Keyword Model
        ├── BufferPluginCreateCfg_2s_64K_960FrameSize_drop_old_q15.bin    #
Buffer Plugin configuration file
        ├── ksp_vp_vui_raf_control.dat                                #
Knowles Vocice Processing Algo config file
        ├── ksp_vp_vui_raf_create.dat                                 #
Knowles Vocice Processing Algo config file
        └── RomeApp.bin                                               #
Knowles DSP Firmware
```

3.2.6 Device Tree Configurations

Knowles's kernel driver requires the following node to be present in the host platform's device tree file. In Raspberry Pi it can be added as an overlay file. create the overlay file as below and modify arch/arm/boot/dts/overlays/Makefile to include the dts file for compiling.

3.2.6.1 RaspberryPi

The device tree overlay file can be found in the source code package under the dts overlay directory: arch/arm/boot/dts/overlay/ia8x01-adnc-overlay.dts

3.2.6.1.1 GPIO fragment

This DTS overlay fragment defined the GPIO pins used by the Knowles ADNC driver for controlling the IA8x01 DSP chip.

Device Tree Overlay: ia8x01-adnc-overlay.dts

```
/dts-v1/;
/plugin/;

/ {
        compatible = "brcm,bcm2835", "brcm,bcm2709", "brcm,bcm2710";

        fragment@1 {
                target = <&gpio>;
                __overlay__ {
                        iaxxx_pins: iaxxx_pins {
                                brcm,pins = <5 6 13>;
                                brcm,function = <1 0 1>;
                        };

                };
        };
```

3.2.6.1.2 SPI Bus Driver fragment

The SPI bus driver is the platform device descriptor for the Knowles ADNC driver. It defines all the major paraments required by the driver to configure correctly and communicated with the underlying DSP hardware.

Device Tree Overlay: ia8x01-adnc-overlay.dts

```
/dts-v1/;
/plugin/;

/ {
    compatible = "brcm,bcm2835", "brcm,bcm2709", "brcm,bcm2710";

    fragment@2 {
        target = <&spi0>;
        __overlay__ {
            status = "okay";
            #address-cells = <1>;
            #size-cells = <0>;

            iaxxxspi0: iaxxx@0 {
                compatible = "knowles,iaxxx-spi";
                reg = <0>;
                spi-max-frequency = <6000000>;
                spi-cpha;
                pinctrl-names = "default";
                pinctrl-0 = <&iaxxx_pins>;
```

```
                adnc,reset-gpio = <&gpio 5 0>;
                adnc,event-gpio = <&gpio 6 0>;

                adnc,spi-sbl-speed = <6000000>;
                adnc,spi-app-speed = <6000000>;

                interrupt-parent = <&gpio>;
                interrupts = <6 1>;
                status = "ok";

                iaxxx_codec0: iaxxx-codec {
                    #sound-dai-cells = <0>;
                    compatible = "adnc, iaxxx-codec";
                    status = "okay";
                };
            };

        };
    };
```

The above node should be added as child-node to device node of host's SPI interface used to connect to Knowles DSP.
The settings in the child-node are mandatory for Knowles kernel driver to boot up properly

3.2.6.2 Driver-name

compatible = knowles,iaxxx-spi;
This setting uniquely identifies the Knowles kernel driver and ensures it is used when the driver is probed for binding to the spi device.

3.2.6.3 SPI Settings

1. SPI default speed
   The setting sets the default SPI speed for the host to initially to communicate with Knowles' Audio DSP after chip reset until it boots up.
   For example:
   For setting default SPI speed to 9.6Mhz: `spi-max-frequency = <9600000>;`
2. SPI normal speed
   This setting sets the SPI speed used after Knowles' Audio DSP has booted up.
   For example:
   For setting normal SPI speed to 9.6Mhz:`adnc, spi-app-speed = <9600000>`

3.2.6.4 GPIO Settings

Knowles DSP expects the following two GPIOs to be setup in host-side

1. Reset
2. Event (interrupt)

The device tree entries "adnc,reset-gpio" and "adnc,event-gpio" are used to setup the corresponding GPIO pins.
For example:

```
adnc,event-gpio = <&gpio5 0>;

adnc,reset-gpio = <&gpio6 1>;
```

3.2.6.5 CODEC Driver name (Optional):

compatible = `knowles, iaxxx-codec`;
So, as per the platform the codec details can be added in DTS or Machine driver.

3.2.7 ALSA Codec Driver Configuration

3.2.7.1 Simple Audio card

IA8x01 codec dai details can be added in either DTS or Machine driver file depending on the platform being used.

In case of Raspberry Pi, the RaspberryPi platform uses "simple-audio-card" framework to register the CPU, platform, codec dais, and also specify the format, clock master.

This data is used by the simple-audio-card framework to create the link between CPU dai, platform dai and codec dai. The simple audio card the dai-link-names, PCM format details from the .dts file (see the following code snippet) and creates a custom sound card for the IA8X01A.

For more information about available simple-audio-card device tree properties, see the following kernel documentation *Documentation/devicetree/bindings/sound/simple-card.txt.*

3.2.7.2 ALSA Sound Card fragment

In the current example, we create a sound card corresponding IA8x01 device with TLV AIC32 Audio DAC is used in the system for the audio playback output. This might change on the customer board.

Device Tree Overlay: ia8x01-adnc-overlay.dts

```
/dts-v1/;
/plugin/;

/ {
        compatible = "brcm,bcm2835", "brcm,bcm2709", "brcm,bcm2710";

    fragment@5 {
        target = <&i2s>;
        __overlay__ {
            status = "okay";
        };
    };

    fragment@6 {
        target = <&sound>;
        sound_overlay: __overlay__ {
            compatible = "simple-audio-card";
            simple-audio-card,format = "i2s";
            simple-audio-card,name = "audio-iaxxx";
            status = "okay";
            #address-cells = <1>;
            #size-cells = <0>;

            simple-audio-card,dai-link@0 {
                reg = <0>;
                format = "i2s";
                cpu {
                    sound-dai = <&i2s 0>;
                };
                codec {
                    sound-dai = <&iaxxx_codec0>;
                };
            };

            simple-audio-card,dai-link@1 {
                reg = <1>;
                format = "i2s";
                cpu {
                    sound-dai = <&i2s 0>;
```

```
                };
                codec {
                    sound-dai = <&tlv320aic32x4>;
                };
            };
        };
    };
```

3.2.8 User space integration

3.2.8.1 Add new source files to customer workspace

The below table lists the files that should be copied into the customer workspace with little or no modification:

The source files/folders provided in the release package have to be copied to the specified path in the target source tree.

Source	Destination	Target after compilation (.so/.jar)
"VQ HAL" Component		
<>\chelsea-iot-middleware\source\knowles\libs	<>\<user-space-libs-paths>\Knowles\	libvqhal.so libktpcm.so libstrm.so
"Voice Processing HAL and Tunnel HAL" Components		
<>\chelsea-iot-middleware\source\knowles\libs	<>\<user-space-libs-paths>\Knowles\	libodsp.so libtunnel.so
Console Applications		
<>\chelsea-iot-middleware\source\knowles\console_apps	<>\console-apps\Knowles\	avs_voice_wake (For AVS integration) voice_wake (simple Voice Wake App) tunneling_hal_test_chelsea (for debugging)
Kernel Headers		
<>\chelsea-iot-middleware\source\knowles\kernel-headers	<>\<kernel-header-path>\	NA
External libraries		
chelsea-iot-middleware\source\external\tinyalsa	<>\<user-space-libs-paths>\external\	libtinyalsa.so
chelsea-iot-middleware\source\external\ avs-device-sdk	-	AVS SDK updated with Knowles changes (To show how Knowles components are added to AVS SDK)

3.2.8.2 AVS Device SDK Integration

We need to integrate the Knowles Wake Word plugin into the AVS Device SDK. The following files are the new files that need to be added into the AVS Device SDK.

- SampleApp/include/SampleApp/KnowlesMicrophoneWrapper.h
- SampleApp/include/SampleApp/KnowlesKeywordObserver.h
- SampleApp/include/SampleApp/sock_msg_utility.h
- SampleApp/src/KnowlesKeywordObserver.cpp
- SampleApp/src/KnowlesMicrophoneWrapper.cpp
- KWD/Knowles/include/Knowles/KnowlesKeywordDetector.h
- KWD/Knowles/include/Knowles/IKnowlesKWEventListener.h
- KWD/Knowles/src/KnowlesKeywordDetector.cpp
- KWD/Knowles/src/CMakeLists.txt
- KWD/Knowles/CMakeLists.txt

The following files were changed to integrate the Knowles Wake Word plugin, please use this as a reference to modify the AVS Device SDK.

- SampleApp/CMakeLists.txt
- SampleApp/src/CMakeLists.txt
- SampleApp/src/SampleApplication.cpp
- SampleApp/src/InteractionManager.cpp
- KWD/CMakeLists.txt
- KWD/KWDProvider/src/CMakeLists.txt
- KWD/KWDProvider/src/KeywordDetectorProvider.cpp
- build/cmake/KeywordDetector.cmake

3.2.8.3 Build Platform

Please export the following variables with the correct location to the toolchain

```
export TOOLCHAIN_PATH=<full-path-to-toolchain>
export TOOLCHAIN_PREFIX=<prefix-for-toolchain>
```

Execute the following command from all the sub-directories to build the source:

```
$ make
```

3.2.8.4 Binaries generated

3.2.8.4.1 Libraries:

The following libraries will be generated. They should be copied into the rootfs directory.

Library name	Description
libvqhal.so	The VQ HAL library which will help in loading the keyword model and performing the keyword recognition.
libktpcm.so	The Knowles Tunneling PCM library, this provides an ALSA like API to get the audio data from the chip
libodsp.so	A library that provides an abstraction over the ODSP IOCTL calls.
libtunnel.so	A library that provides an abstraction over the Tunnel IOCTL calls.
libtinyalsa.so	A library that provides ALSA kcontrols to interact with the codec driver.
libstrm.so	A library that provides streaming API's for the AVS

3.2.8.5 Console Applications:

The following console applications are generated, which must be copied to the rootfs directory.

Console application name	Description
avs_voice_wake	A utility to connect to the AVS SDK. This will open the socket connection to the AVS SDK and will be responsible for communicating with the AVS SDK.
voice_wake	A utility that can be used to test the voice recognition.
tunneling_hal_test_chelsea	A simple tunneling utility which tunnels out data from the given list of source end points.

3.2.8.6 Running the console applications

This sections describes on how each application works and what to expect.

3.2.8.6.1 Simple voice wake app

A utility that can used to test the voice recognition. This simple application shows how to use the VQ HAL library and how to interact with the library to start the keyword recognition and to get the keyword detection events. It also shows how to extract the audio after the keyword detection has occurred. The audio file is saved in the location /data/data,/ please create this directory if it is not present on the device.

Usage -

$ mkdir -p /data/data

$ voice_wake

3.2.8.6.2 AVS Voice Wake

A utility that connects to the AVS Device SDK via socket. This utility is responsible for connecting to the AVS Device SDK and for interacting with the Voice Wake library. When the keyword is detected it is forwarded to the AVS Device SDK via the socket and listen for Audio Stream communication from the AVS Device SDK.

Usage -

$ avs_voice_wake
$ avs_voice_wake –s
In the first form, it enables the connection to the AVS socket.

In the second form, it creates a socket server connection to which an AVS client can join.

3.2.8.6.3 Tunneling_hal_test_chelsea

This utility can be used to tunnel out data from single/multiple end points at any point of time, as long as the route is enabled. This utility can be run in multiple shells at the same time. The output files are located in the dir /data/data/ and with the filename prefix - "tnl_op"

Usage -

$tunneling_hal_test <instance number> <Number of tunnels> <Time in seconds> <Source End pt 1> <tnl mode> <encode fmt> <Source End pt 2> <tnl mode> <encode fmt>...
Description -

<instance number> - The instance number of the current utility, this number will be used in the file name of the output from this utility and helps in differentiating outputs from different utilities.

<Number of Tunnels> - Total number of end points that need to be tunneled out

<Time in seconds> - Time period in seconds for which this utility needs to run for. Giving 0 as input will make this utility run forever or until <ctrl-c> is pressed

<Source End pt1> - Source end point number, this is usually defined in the route diagrams

<tnl mode> - Tunnel modes

Type	Value
Synchronous	0
Asynchronous	1

-

<encode format>: Tunnel encoding formats.

Type	Value
Opaque	0
Afloat	1
Q15	0xF

3.2.9 Basic integration validation

The first thing to verify after successful software integrations is to check the Linux Kernel logs (dmesg) and confirm the below items are proper.

- Check the kernel logs to make sure the Knowles ADNC driver is getting probed.

```
root@raspberrypi:/home/pi# dmesg | grep -i iaxx

[   18.701655] iaxxx_mfd_core: loading out-of-tree module taints
kernel.
[   18.714265] iaxxx-spi spi0.0: SPI Device Id 0
```

- Make sure the Knowles DSP firmware is getting downloaded successfully. This will ensure that the DTS entries, Bus configuration and the Firmware Binaries are proper and in place.

```
[   19.911869] iaxxx-spi spi0.0: FW boot complete event
iaxxx_event_handler: src:0x2a10
[   19.911919] iaxxx-spi spi0.0: Confirm switch to App mode
[   19.911949] iaxxx-spi spi0.0: Firmware running in application
mode
```

- Check the ANDC codec and sound card registration is successful

```
root@raspberrypi:/home/pi# cat /proc/asound/cards
 0 [Headphones     ]: bcm2835_headphonbcm2835 Headphones -
bcm2835 Headphones
                      bcm2835 Headphones
 1 [audioiaxxx     ]: audio-iaxxx - audio-iaxxx
                      audio-iaxxx
```

The kernel driver integration is successful if all of the above steps are proper. Now we can run a minimal audio route to check the basic DSP features are working from a kernel console level.

3.2.9.1 Validation using console Application

We can try running a basic console application to verify the basic audio routing including voice wake support is working.

The simple voice_wake reference application sets up a voice wake route to detect the keyword. It will automatically start collecting 5seconds of buffer audio data after a successful keyword detection and save it into a file.

This step will ensure that all the software components are integrated successfully into the Host system.

`> voice wake #` For usage and other information, see section S*imple voice wake app*.

The expected output from the voice_wake application is that it prints console logs when the keyword is detected and the audio streaming is completed successfully.

```
IAXXX_VQ_EVENT                                                    # Keyword
Detection Logs
Current state START
+iaxxx_odsp_evt_getevent+
iaxxx_odsp_evt_getevent: dev id 0, event id 0, data 1
-iaxxx_odsp_evt_getevent-
Eventid received is EVENT_ID_KW_ID
New state EVENT_KW_DETECT
[...]

amazon_start_audio: Streaming from amazon buffer plugin end point    # Starting
to dump the audio stream
kt_pcm_open: Start frame = 486
Entering ia_start_tunneling
Entering ia_enable_tunneling_source
[...]

Entering ia_stop_tunneling                                          # Completed
saving the audio stream to the file
Total Frames repeated 0
Total Frames dropped 0
```

3.2.10 Debugging

3.2.10.1 Linux Kernel Logs

IA8X01 Linux driver prints all the logs to the dmesg with the prefix "iaxxx and ia8x01". Search for "iaxxx and ia8x01" in the kernel log to check all relevant logs for Chelsea kernel log should print any errors and crash detected by the driver and if any recovery triggered. Also the corresponding device id will be visible on the dmesg which can be used to identify the device generating the logs

3.2.10.2 Commands to read FW, plugin and package version

There are many sysfs entries exposed by the ADNC driver to fetch the module version strings from the DSP chip.

```
# Read the Knowles DSP Firmware version String
cat /sys/bus/spi/devices/<SPI device>/iaxxx/fw_version

# Read the DSP Algorithm Plugin version
cat /sys/bus/spi/devices/<SPI device>/iaxxx/plugin_version

# Read the DSP Algorithm Package verssion
cat /sys/bus/spi/devices/<SPI device>/iaxxx/package_version
```

3.2.10.3 Crash and Debug Information

The crash/debug log infrastructure is used to analyze issues related to the Rome firmware or Algorithm running on the DSP chip. The Host driver implements a mechanism to read this information automatically and keep it a kernel driver memory for the users to dump it into a file and analyze it using Knowles Software tools

The crash logs will be collected whenever a crash happens. It will be available in the kernel memory until the next crash happen. During the next crash, it will be overwritten.

```
# Read the DSP debug log information
cat /dev/debug_log0

# Read the DSP crash information if there was a crash occured
cat /dev/crashdump0 > crash_log.bin
```

Chapter 4: Customisations

4.1 Overview

The Reference IA8201 Voice Wake Solution can be customized to suit the customer's product needs.

Following customizations can be done on top of the Reference IA8201 Voice Wake Solution:

- Changing the Wake Word algorithm
- Use a new Wake Word Model instead of the default "Alexa" KW
- Change the Mic spacing/geometry in both 2-Mic & 3-Mic cases
- Use a different Linux Host than Raspberry Pi

The following sections provide instructions on how to achieve these customizations

4.2 Changing the Wake Word Algorithm

The Reference IA8201 Voice Wake Solution supports 3 Voice Wake algo options: Amazon VT, Retune/VoiceSeeker & Sensory.

This section explains how to choose different Wake Word Algorithm from the SW stack and build it for the customer platform

4.2.1 Supported Wake Words Algorithm Plugin in the SW package.

As described above, the SW package supports 3 wake word algorithms and the default one will be Amazon Wake Word algorithm.

VQ HAL wake word algorithms plugins

```
vq_hal_algorithm_plugins/              # VQ HAL Algorithm plugins (Amazon, Retune and Sensor
|
├── amazon_vt_plugin.c                 # Amazon Wake Word plugin
|
├── knvp_sensory_vt_plugin.c           # Sensory Wake Word plugin
|
└── retune_vq_plugin.c                 # Retune Wake Word plugin
```

For each algorithm plugin to work, it needs the corresponding DSP side binary files (Algorithm binary, mic configuration, wake word files and IA8x01 Firmware in some cases) from the vendor. The default SW package only ships the DSP side binaries for the Amazon Wake Word Algorithm. For Sensory & Retune Algos, customer needs to get the Add-on packages from the Algo vendor.

A typical algorithm package contains the below item (ex: Amazon VW).

wake word algorithm package contents

```
└── audience
      └── ia8x01
          ├── AmazonWWCreateConfig.bin                  # Algorithm Create config
          ├── AmazonWWPackage.bin                       # Algorithm Plugin package
          ├── WR_250k.en-US.alexa.bin                   # Wake Word Model File
          |
```

```
├── ksp_vp_vui_raf_control.dat                          # MIC topology configuration
├── ksp_vp_vui_raf_create.dat                           # MIC topology configuration
│
├── RomeApp.bin                                         # Rome Firmware Binary
└── BufferPluginCreateCfg_2s_64K_960FrameSize_drop_old_q15.bin  # Audio
                                                        stream Buffer configuration
```

4.2.2 Steps to Change the Wake Word Algorithm Plugin

1. Copy the corresponding algorithm plugin file from "vq_hal_algorithm_plugins" source package directory into the "VQ HAL" modules directory <>\chelsea-iot-middleware\source\knowles\libs\src\algo_plugin.c. Make sure that file is renamed to "algo_plugin.c".
2. Build the VQ HAL libraries and copy it to location "/usr/lib" on the host.
3. Get the Algorithm Add-on package and copy it under the "/lib/firmware/audience/" directory on the host. Make sure to name the file as per the macros defined in the algo_plugin.c

4.3 Changing the Wake Word Model

The "Reference IA8201 Voice Wake Solution" supports 3 Voice Wake algo options : Amazon VT, Retune/VoiceSeeker & Sensory.

When using Retune/VoiceSeeker & Sensory VT algos, it is possible to use a different Wake Word Model supported by the algorithm vendors.

This section explains how to integrate a new Wake Word Model into the SW stack.

4.3.1 The default Wake Word

The Wake Word model file will be available as a binary file from the Algorithm vendor and it is placed under the "/lib/firmware/audience/" directory for the HAL to download it when the audio route is setup.

Please find the vendor-specific algorithm binary files located under the "firmware" and the file "WR_250k.en-US.alexa.bin" is corresponding to the Alexa US English Keyword Model file.

wake word model file

```
/lib/firmware/
└── audience
    └── ia8x01
        ├── WR_250k.en-US.alexa.bin                     # Amazon Alexa Keyword Model
```

The Keyword model file is downloaded by the corresponding "algo_plugin.c" for each algorithm. The keyword model macro is defined in the "VQ HAL" Component "<>\chelsea-iot-middleware\source\knowles\libs\src\algo_plugin.c. Please refer to below mentioned macro definitions for different algorithms.

4.3.1.1 Amazon VW

amazon_vt_plugin.c

```
#define VW_AMAZON_CREATE_DAT_FILE  "audience/ia8x01/WR_250k.en-US.alexa.bin"
```

4.3.1.2 Sensory VW

knvp_sensory_vt_plugin.c

```
#define VW_SENSORY_CREATE_DAT_FILE  "audience/ia8x01/Alexa_pc60_250KB_op3_merged.bin"
```

4.3.1.3 Retune VW

retune_vq_plugin.c

```
#define ALEXA_KW_MODEL                "audience/ia8x01/Retune_Alexa_model.bin"
```

4.3.2 Steps to Update the Wake Word

1. Obtain the Wake Word model file from the Algorithm vendor. Make sure it is compatible with the Wake Word Library version on the system.
2. Copy it into the host system path "/lib/firmware/audience/" and make to rename it accordingly if required.
3. If the Wake Word file name is different from what is mentioned in the default SW package, please update the "wake word file" macro defined in the"algo_plugin.c" and recompile the VQ HAL libs

Please make sure that the wake word file and the libs are copied to the host system before rebooting the system for it to take effect.

4.4 Changing the Microphone Topology

The "Reference IA8201 Voice Wake Solution" supports below Mic configurations:

- 2-Mic : Supported with Amaon VT, Retune/VoiceSeeker & Sensory VT algo options

In case of Amazon VT & Sensory VT, the Front End Pre-processing algo is provided by Knowles. So Knowles is responsible for Mic spacing configuration

- 3-Mic : Supported with Retune/VoiceSeeker algo option

In each of the 2-Mic & 3-Mic configurations it is possible to change the Mic spacing/geometry as per the product needs.

Based on the algorithm option, customers need to get the updated configuration file from the corresponding algorithm vendor for the selected Mic spacing/geometry

This section explains how to integrate a new Mic spacing/geometry configuration file into the SW stack.

4.4.1 MIC Topology Information

The Microphone geometry/spacing information is passed to the Algorithm for getting a better performance output. This information will be provided as a binary file(s) along with the Algorithm library.

It will be downloaded by the HAL during the audio route set up along with the Algorithm binary. The MIC topology binary and how it is passed to the algorithm plugin on DSP will vary from one algorithm vendor to the other.

The default SW package with Knowles Voice Processing and Amazon Wake Word has the below two files to specify the MIC topology (spacing/geometry) information.

4.4.1.1 Knowles VP + Amazon VW or Sensory VW

[Knowles VP + Sensory VT] also has the same MIC topology information as the front end algorithm is common for both Sensory & Amazon VT algorithm packages.

```
/lib/firmware/
└── audience
    └── ia8x01
        ├── ksp_vp_vui_raf_control.dat       # Knowles Voice Processing Algo Turning
paramets
        ├── ksp_vp_vui_raf_create.dat        #
```

Macro definition for the files

file name macros
#define VP_CREATE_DAT_FILE "audience/ia8x01/ksp_vp_vui_raf_create.dat"
#define VP_CONTROL_DAT_FILE "audience/ia8x01/ksp_vp_vui_raf_control.dat"

4.4.1.2 Retune VW

For the Retune Algorithm, the MIC topology information is passed as the below binary files.

```
audience/
└── ia8x01
    ├── LinearMicArray25MMCreate2Mic.bin       # 2MIC 25mm mic array config
    ├── LinearMicArray25MMCreate.bin           # 3MIC 25mm mic array config
```

Macro definition for the files

```
// 3MIC configuration
#define LINEAR_MIC_ARRAY_25MM_CC      "audience/ia8x01/LinearMicArray25MMCreate.bin"

// 2MIC configuration
#define LINEAR_MIC_ARRAY_25MM_2MIC_CC "audience/ia8x01/LinearMicArray25MMCreate2Mic.bin"
```

4.4.2 Steps to Change MIC Topology

1. Obtain the correct mic topology information from the corresponding algorithm vendor
2. Copy the file to the host system "/lib/firmware/audience/" and make to rename it accordingly if required.
3. If the file names are different from what is mentioned in the default SW package, please update the corresponding macro defined in the "algo_plugin.c" and recompile the VQ HAL libs

4.5 Integrating into different Linux Host Platforms

The "Reference IA8201 Voice Wake Solution" running on the EVM Kit uses Raspberry Pi running Linux OS as the Host processor. However, it is anticipated that customers may use some other Linux based Host processor.

Since the Host SW stack is based on standard Linux OS frameworks & primitives, we expect the reference Host SW stack to run on any other Linux Host processor without significant changes.

This section explains key things to handle when porting to a different Linux Host Platform

4.5.1 Kernel Driver Building Changes

The Knowles Driver build steps are explained in the integrations section as it fits under any Linux based Host build systems without any change.

To statically Build the driver into the Kernel Image, use the Kconfig and Defconfig files provided as part of the SW package and copy it to the specified location. We need to update the correct Host Processor Cross compiler toolchain details in the Makefile and trigger the build. Please make sure the Firmware binaries are copied to the right firmware folder inside the target filesystem

4.5.2 Device Tree Changes

The SW package contains a device tree overlay file which contains all the Knowles Driver device information. The Device Tree Overlay Blob information has to be passed to the Kernel Via the bootloader using Host processor supported mechanism. Raspberry Platform uses a "config.txt" file passed by the bootloader to specify the DT overlay blob. But this may vary based on the Host platform Customer is using.

Some of the Host platforms may not support the device tree overlay files directly. In that case, all the device tree nodes required by the Knowles ADNC driver have to be copied accordingly under the child node of the Host systems device tree file.

4.5.3 ALSA Machine Driver and Sound Card Registration

The SW package uses the standard "simple-audio-card" machine driver implementation provided by the ASoC layer to register the sound card with the Knowles codec driver and CPU dai interface.

This might vary according to the Host Platform as they may be using dedicated machine drivers to register the sound cards. The Customer has to take the DAI links details from the sound card DTS fragment and update it in the customer machine driver to complete the sound card registration. Please note that the sound card name is important as it is used by the HAL layerit to automatically find out the card number to execute the mixer controls to setup audio routes.

4.5.4 Middleware changes

Please export the following variables with the correct location to the toolchain and rebuild the middleware code for the desired host platform.

```
export TOOLCHAIN_PATH=<full-path-to-toolchain>

export TOOLCHAIN_PREFIX=<prefix-for-toolchain>
```

Revision History

Version	Comments	Date
0.1	Ported to Knowles template formatted and edited.	06/23/2021

Knowles Corporation
1151 Maplewood Drive
Itasca, Illinois 60143

Phone: (630) 250-5100
Fax: (630) 250-0575
info@knowles.com

Model/Reference Number:
AUD-ESP-00518, Rev 0.1