PSOC™ Edge E84 MCU Voice & Audio Firmware Components Guide

About this document

Scope and purpose

This application note contains details of all voice and audio firmware components that can be used in the audio pipeline on the PSOC™ Edge MCU family. It helps the developers to understand the purpose of every voice and audio firmware component. This application note also explains to the developers on how to use audio and voice components so that they can be modified according to audio application requirements.

Intended audience

This document is intended for users developing and customizing audio applications on PSOC™ Edge MCU.

Table of contents

About this document . . . 1
Table of contents . . . 2
1 Introduction . . . 4
2 Audio Input - Microphones . . . 5
2.1 Pulse Density Modulation (PDM) Digital Microphone . . . 5
2.1.1 PDM mic and PDM-PCM IP block . . . 5
2.1.2 PDM-PCM Peripheral Driver Library(PDL) API . . . 6
2.1.3 PDM-PCM Application usage . . . 7
2.1.3.1 PDM-PCM Configuration . . . 7
2.1.3.2 Application flow . . . 12
3 DEEPCRAFT™ Audio Enhancement . . . 16
3.1 Overview . . . 16
4 High Performance Core Inference Libraries . . . 17
4.1 Overview . . . 17
4.2 DEEPCRAFT™ Voice Assistant . . . 17
4.2.1 Description . . . 17
4.2.2 Training Voice Assistant Models . . . 17
4.2.3 Integration to code example . . . 20
5 Audio Software Codec . . . 21
5.1 Software Codecs Overview . . . 21
5.2 Types of Software Codecs . . . 21
5.2.1 SBC codec . . . 21
5.2.2 Opus codec . . . 22
5.2.3 LC3 codec . . . 22
5.2.4 MP3 codec . . . 23
5.2.5 AAC codec . . . 23
5.3 Code Flow . . . 24
6 Bluetooth® /Bluetooth® Low Energy (BT/BLE) for Audio & Voice . . . 27
6.1 Introduction to Bluetooth® for Audio & Voice Applications . . . 27
6.1.1 System architecture . . . 27
6.1.2 Bluetooth® stack . . . 28
6.1.3 Bluetooth® profiles for Audio & Voice Applications . . . 29
6.2 PSOC™ Edge E84 Bluetooth® Audio & Voice Use-Cases . . . 32
6.2.1 Bluetooth® Classic - A2DP Source . . . 32
6.2.2 Bluetooth® Classic - A2DP Sink . . . 35
6.2.3 Bluetooth® Classic - HFP . . . 36
6.2.4 Bluetooth® LE Audio - Unicast Sink . . . 38

Table of contents

Revision history 41

Trademarks 42

Disclaimer 43

1 Introduction

PSOC™ Edge MCU can be used for various audio & voice use-cases. The audio pipeline for a typical application consists of various components such as audio capture, processing of audio data, ML inference and playback.

For enabling developers to quickly develop their use-case applications, there are various voice & audio firmware components as part of PSOC™ Edge MCU software.

The below picture shows the available voice & audio firmware components which developers can use. Most of the components have related Code Examples available in https://github.com/Infineon/

Users can refer to the code examples and adapt them to the use-cases.

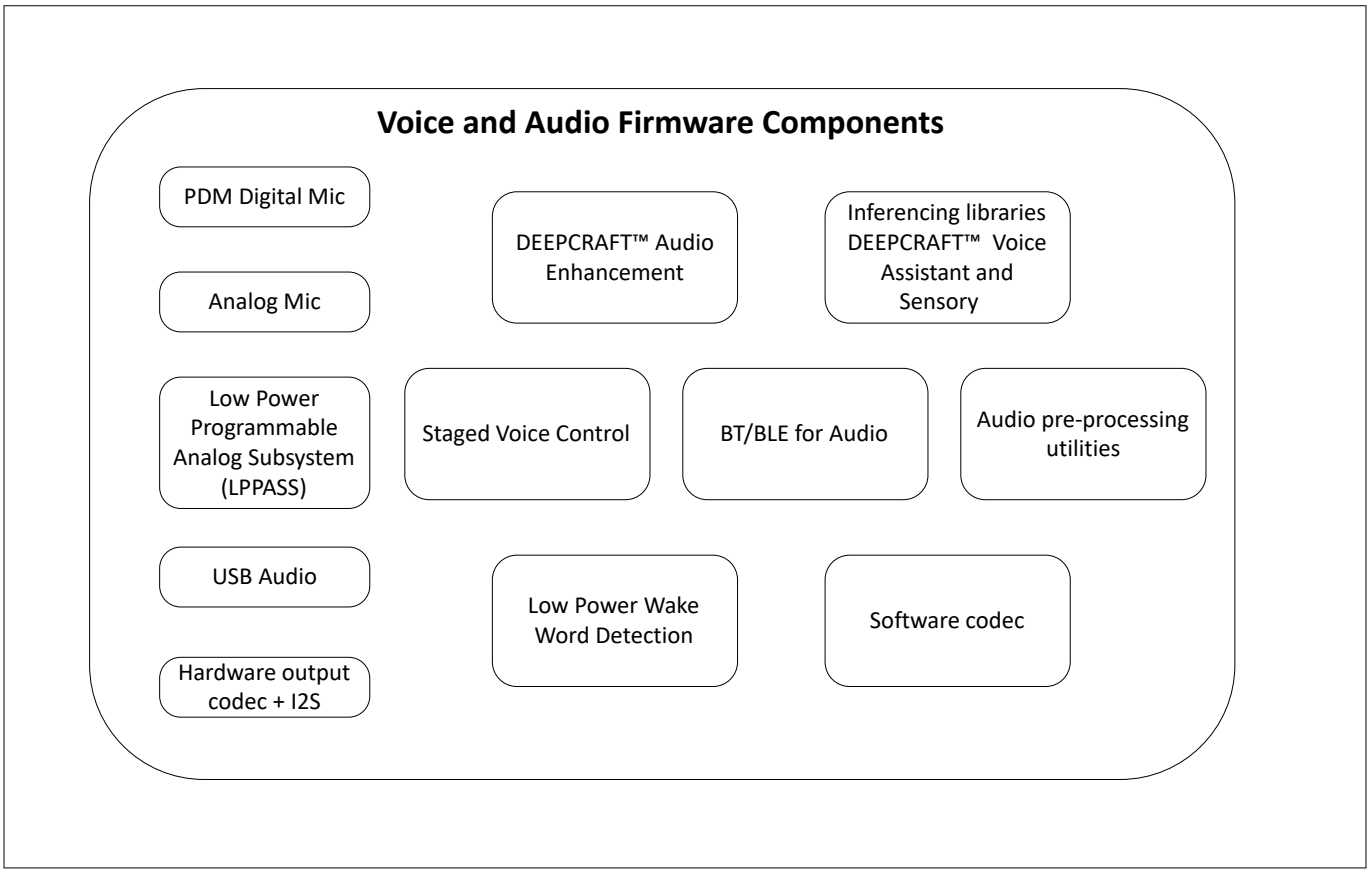


Figure 1 Voice and Audio Firmware Components

In this application note, each of the component is briefly explained with references to code-examples and other useful details for application developers.

2 Audio Input - Microphones

PSOC™ Edge MCU acquires audio data via microphones. There are two types of microphones on-board the PSOC™ Edge MCU kit. They are

- Digital mic - Pulse Density Modulation (PDM) mic
- Analog mic

***Note**:* *There is also another option to stream audio from PC to the kit via USB Audio Class. We do not cover USB audio in this chapter.*

The PDM mic is generally used for mains powered applications where low power is not a mandatory feature. Analog mic is recommended in use cases that require low power.

2.1 Pulse Density Modulation (PDM) Digital Microphone

This chapter briefly explains about using PDM mic on PSOC™ Edge MCU. The related Code Examples which users can refer to are

- PDM-PCM Code Example - https://github.com/Infineon/mtb-example-psoc-edge-pdm-pcm This is a basic code example that demonstrates capturing audio data with PDM mic
- DEEPCRAFT™ Audio Enhancement Code Example - https://github.com/Infineon/mtb-example-psoc-edge-ae-application This code example demonstrates using PDM mic with further processing of audio data
- Mains Powered Local Voice Code Example - https://github.com/Infineon/mtb-example-psoc-edge-mains-powered-local-voice This code example demonstrates using PDM mic for actual end-user use-cases with complete audio pipeline

2.1.1 PDM mic and PDM-PCM IP block

Pulse Density Modulation (PDM) is a digital modulation technique used to encode analog signals into digital format. It is particularly useful for representing audio signals in digital audio systems.

In PDM, the information is carried by the density or rate of pulses. The PDM technique is simple and very efficient. It is relatively easy to implement using basic digital circuits, and the process of encoding and decoding PDM signals can be performed with minimal computational complexity.

PDM is commonly used in applications where high-resolution and high-fidelity audio is required.

The PSOC™ Edge MCU has a PDM-PCM converter IP block to convert the PDM output from microphones to Pulse-Code Modulation (PCM) data.

A total of six PDM microphones can be interfaced to PSOC™ Edge MCU.

A PDM microphone, also called a digital microphone, consists of the following parts:

- A microphone element (typically, this is an electret capsule)
- An analog preamplifier
- A PDM modulator
- Interface logic

Users can interface any suitable PDM mic to the kit.

The analog signal from the microphone element is first amplified, then sampled at a high rate and quantized in the PDM modulator. The modulator combines the operations of quantization and noise shaping. The output is single bit at a very high sampling rate. The noise shaping operation ensures that the noise in the audio band is relatively low, while the noise above the audio band is relatively high.

PSOC™ Edge MCU contains a PDM-PCM converter IP block that converts PDM input to PCM output at the hardware level without any CPU intervention. The PSOC™ Edge MCU requires PDM-PCM conversion because the

software processes the audio data as raw PCM. This enables easier processing of data as it is serialized and can also be stored in memory/file for offline analysis.

The PDM to PCM conversion happens by sampling the PDM signal at a lower sampling rate (decimating) and increasing the word length. The ratio of the PDM bit rate to the decimated PCM sampling rate is called the oversampling ratio.

The sample rate needs to be reduced by the oversampling ratio. This is accomplished by a digital filtering operation called decimation. Decimation is the counterpart to interpolation, that is, samples are removed from the signal to reduce the sampling rate. It is important that the noise above the audio band in the 1-bit representation is not allowed to alias into the audio band. The decimation filters are designed to filter out this noise, leaving the baseband audio signal intact. The output of the decimator is a PCM audio stream at the baseband (non-oversampled) rate.

The PDM-PCM converter in the PSOC™ Edge MCU supports both mono and stereo configurations. It has multiple filter options to reduce noise and achieve the required decimation. Also, it supports programmable PCM sample sizes as well.

The following are the supported filters:

- Cascaded Integrator Comb (CIC) filter: Used to reduce the sampling rate of the PDM signal. The PSOC™ Edge MCU has one CIC filter
- Finite Impulse Response (FIR) filter: Used for decimation. The PSOC™ Edge MCU has two FIR filters
- DC blocking filter: Removes the DC component from the FIR filter outputs. The PSOC™ Edge MCU has one DC blocking filter

The high-level flow of the PDM to PCM conversion block with its filters is shown in the following figure.

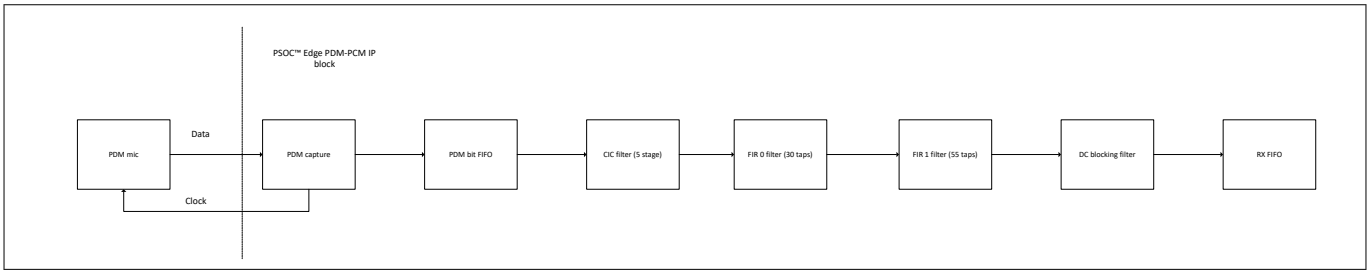


Figure 2 PDM-PCM IP block

The PDM-PCM conversion process is entirely done in hardware, and the PDM-PCM converter simplifies the conversion process and contains different registers that can be configured based on application needs.

For more information on the PDM-PCM converter, refer to the PSOC™ Edge MCU reference manual.

2.1.2 PDM-PCM Peripheral Driver Library(PDL) API

From application code, to control the PDM-PCM IP block, the Peripheral Driver Library (PDL) API is to be used.

The full list of PDL API for PDM-PCM block can be found at https://infineon.github.io/mtb-dsl-pse8xxgp/html/group__group__pdm__pcm__functions__v2.html

Some of the key API that is used in the code examples are listed below

Table 1 Table 1 : PDM-PCM PDL API

API	Description
Cy_PDM_PCM_Init	Initializes PDM-PCM block
Cy_PDM_PCM_Channel_Enable	Enable the PDM-PCM channel (Channel 2 and 3 are used in PSOC™ Edge)

(table continues...)

Table 1 (continued) Table 1 : PDM-PCM PDL API

API	Description
Cy_PDM_PCM_Channel_Init	Initialize the PDM-PCM channel
Cy_PDM_PCM_Channel_ClearInterrupt and Cy_PDM_PCM_Channel_SetInterruptMask	Clear and Set interrupt
Cy_PDM_PCM_Activate_Channel	Activate PDM-PCM channel
Cy_PDM_PCM_SetGain	Set PDM-PCM gain (FIR 1 scale). Ensure to use convert_db_to_pdm_scale() and set_pdm_pcm_gain() to convert dB to scale value.
Cy_PDM_PCM_Channel_ReadFifo	Read PCM data from FIFO
Cy_PDM_PCM_ApplyPCM_Gain	Apply Software gain to PCM samples. Used when PDM-PCM channels are configured to 24 bit word size and converts 24 bit to 16 bit for further processing in the audio pipeline

For the usage of the API, refer to

- https://github.com/Infineon/mtb-example-psoc-edge-pdm-pcm For basic usage with 16 bit word size
- https://github.com/Infineon/mtb-example-psoc-edge-ae-application For actual use-cases and 24 bit word size + software gain

2.1.3 PDM-PCM Application usage

In this section, steps on how to use PDM-PCM block in application is shown. There are two options that can be used.

- PDM-PCM with 16 bit word size - This is the basic configuration with limited gain range
- PDM-PCM with 24 bit word size - In this configuration the usage gain range is larger. The audio pipeline still works with 16-bit data and the conversion from 24-bit to 16-bit happens in the application via PDL API

2.1.3.1 PDM-PCM Configuration

The configuration that is available in the default BSP configures PDM-PCM channels to a word size of 16 bits.

Refer to the https://github.com/Infineon/mtb-example-psoc-edge-pdm-pcm Code Example. In this code example, open design.modus via Device Configuartor. The design.modus is available in

\bsps\TARGET_APP_KIT_PSE84_EVAL_EPC2\config (for EPC2)

or

\bsps\TARGET_APP_KIT_PSE84_EVAL_EPC4\config (for EPC4)

after importing the application.

The below picture shows the default configuration

2 Audio Input - Microphones

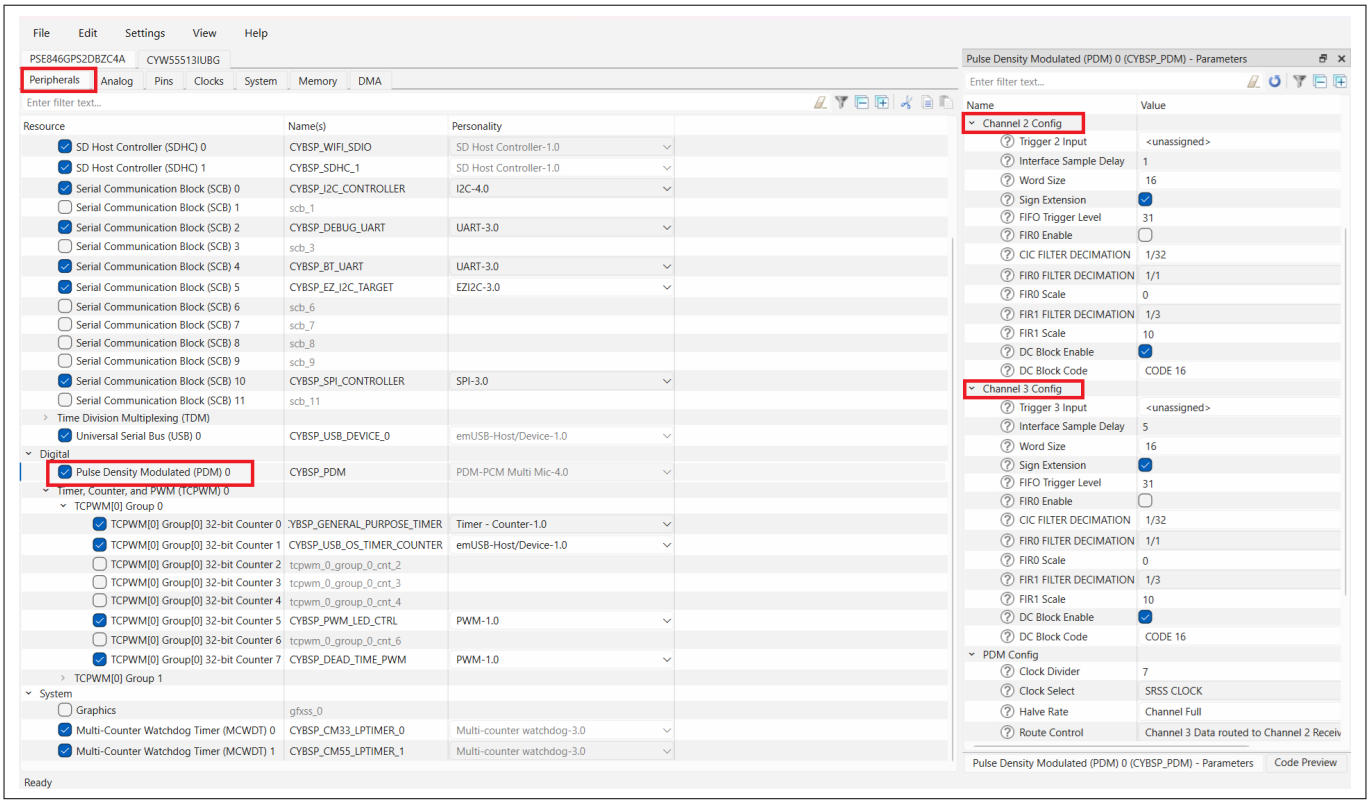


Figure 3 Default PDM-PCM configuration

In PSOC™ Edge MCU, Channel 2 and Channel 3 are used.

The FIR1 Scale value is the gain that is set from the application. Since the PDL API for setting gain can take only scale values, application needs to convert gain in dB to scale value.

The scale values and the corresponding dB is available in https://infineon.github.io/mtb-dsl-pse8xxgp/html/group__group__pdm__pcm__enums__v2.html#ga13d3423cdf014e5cc4828db95914edb4

The below picture shows the same enums as in above link

```
/** PDM Gain Enums */
typedef enum
{
    CY_PDM_PCM_SEL_GAIN_83DB              = 0U,  /**< Gain of 83db corresponding to scale 0 */
    CY_PDM_PCM_SEL_GAIN_77DB              = 1U,  /**< Gain of 77db corresponding to scale 1 */
    CY_PDM_PCM_SEL_GAIN_71DB              = 2U,  /**< Gain of 71db corresponding to scale 2 */
    CY_PDM_PCM_SEL_GAIN_65DB              = 3U,  /**< Gain of 65db corresponding to scale 3 */
    CY_PDM_PCM_SEL_GAIN_59DB              = 4U,  /**< Gain of 59db corresponding to scale 4 */
    CY_PDM_PCM_SEL_GAIN_53DB              = 5U,  /**< Gain of 53db corresponding to scale 5 */
    CY_PDM_PCM_SEL_GAIN_47DB              = 6U,  /**< Gain of 47db corresponding to scale 6 */
    CY_PDM_PCM_SEL_GAIN_41DB              = 7U,  /**< Gain of 41db corresponding to scale 7 */
    CY_PDM_PCM_SEL_GAIN_35DB              = 8U,  /**< Gain of 35db corresponding to scale 8 */
    CY_PDM_PCM_SEL_GAIN_29DB              = 9U,  /**< Gain of 29db corresponding to scale 9 */
    CY_PDM_PCM_SEL_GAIN_23DB              = 10U, /**< Gain of 23db corresponding to scale 10 */
    CY_PDM_PCM_SEL_GAIN_17DB              = 11U, /**< Gain of 17db corresponding to scale 11 */
    CY_PDM_PCM_SEL_GAIN_11DB              = 12U, /**< Gain of 11db corresponding to scale 12 */
    CY_PDM_PCM_SEL_GAIN_5DB               = 13U, /**< Gain of 5db corresponding to scale 13 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_1DB      = 14U, /**< Gain of -1db corresponding to scale 14 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_7DB      = 15U, /**< Gain of -7db corresponding to scale 15 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_13DB     = 16U, /**< Gain of -13db corresponding to scale 16 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_19DB     = 17U, /**< Gain of -19db corresponding to scale 17 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_25DB     = 18U, /**< Gain of -25db corresponding to scale 18 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_31DB     = 19U, /**< Gain of -31db corresponding to scale 19 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_37DB     = 20U, /**< Gain of -37db corresponding to scale 20 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_43DB     = 21U, /**< Gain of -43db corresponding to scale 21 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_49DB     = 22U, /**< Gain of -49db corresponding to scale 22 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_55DB     = 23U, /**< Gain of -55db corresponding to scale 23 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_61DB     = 24U, /**< Gain of -61db corresponding to scale 24 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_67DB     = 25U, /**< Gain of -67db corresponding to scale 25 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_73DB     = 26U, /**< Gain of -73db corresponding to scale 26 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_79DB     = 27U, /**< Gain of -79db corresponding to scale 27 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_85DB     = 28U, /**< Gain of -85db corresponding to scale 28 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_91DB     = 29U, /**< Gain of -91db corresponding to scale 29 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_97DB     = 30U, /**< Gain of -97db corresponding to scale 30 */
    CY_PDM_PCM_SEL_GAIN_NEGATIVE_103DB    = 31U  /**< Gain of -103db corresponding to scale 31 */
} cy_en_pdm_pcm_gain_sel_t;
```

Figure 4 Gain to Scale

For the dB to scale conversion refer to the convert_db_to_pdm_scale() and set_pdm_pcm_gain() API available in the code examples. All the code examples use these API to set the gain within the PDM-PCM block, for eg) Refer to https://github.com/Infineon/mtb-example-psoc-edge-pdm-pcm/blob/master/proj_cm33_ns/source/app_pdm_pcm.c

For applications that require no gain control, or if use case can be achieved with limited gain range, the default configuration is sufficient.

***Note**: With the default configuration, setting gain value greater than 29 dB will cause clipping of audio data in left channel. This is due to the PDM mic component that has unequal DC offset in left and right channels. The PDM mic component used in the kits are IM7* series of XENSIV™ MEMS microphone*

For applications that require gain control from application/dynamic changes to gain, for eg) applications that use DEEPCRAFT™ Audio Enhancement or Inferencing, it is suggested to use 24 bit configuration that has a wider gain range.

The following code examples use PDM-PCM with 24 bit configuration as they demonstrate use-cases and require dynamic control of gain.

- https://github.com/Infineon/mtb-example-psoc-edge-ae-application
- https://github.com/Infineon/mtb-example-psoc-edge-mains-powered-local-voice

Refer to https://github.com/Infineon/mtb-example-psoc-edge-ae-application for the 24 bit usage along with dynamic control of gain via Audio Front End (AFE) Configurator.

2 Audio Input - Microphones

Open the design.modus via device configurator
The design.modus is available in
\bsps\TARGET_APP_KIT_PSE84_EVAL_EPC2\config (for EPC2)
or
\bsps\TARGET_APP_KIT_PSE84_EVAL_EPC4\config (for EPC4)
after importing the application.
The below picture shows the PDM-PCM configuration for 24 bit word size,

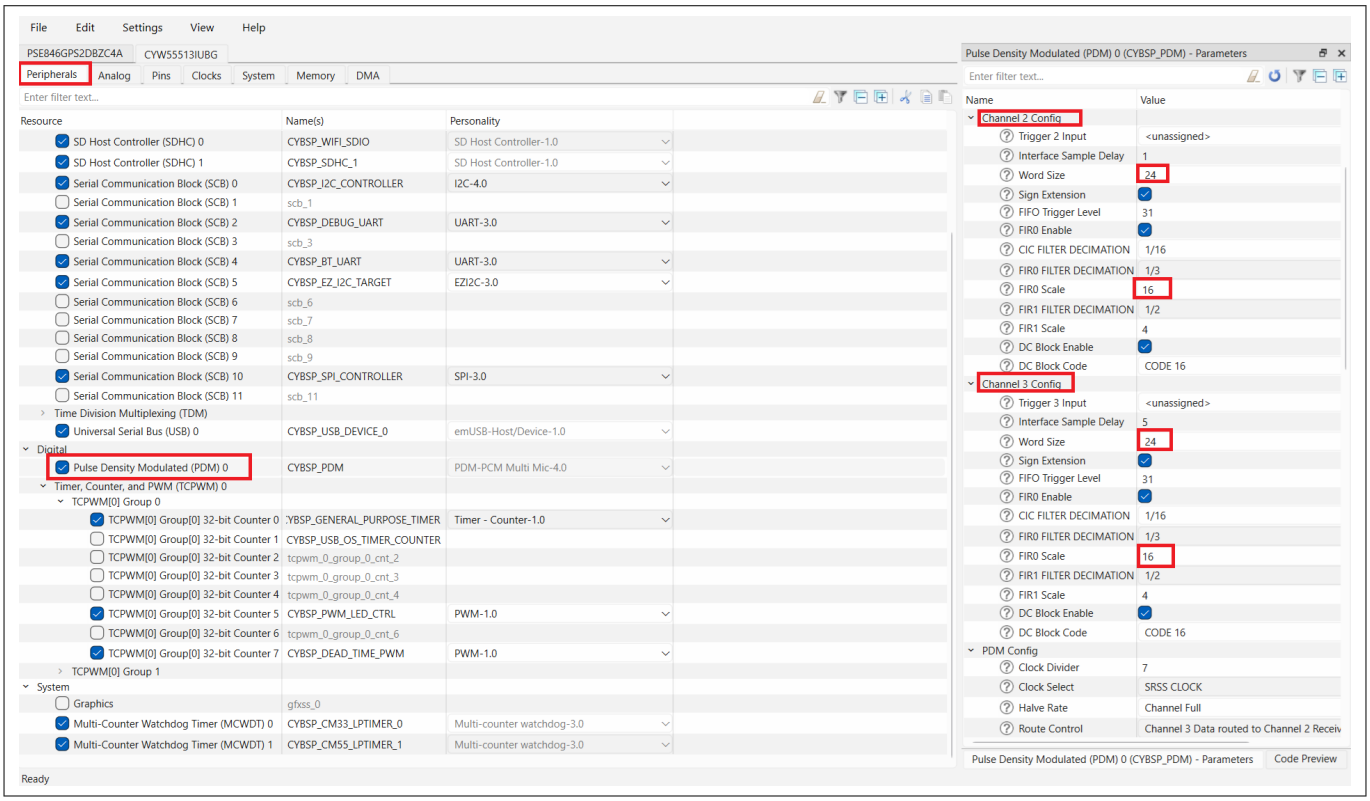


Figure 5 24 bit PDM-PCM configuration

This configuration also requires custom FIR filter coefficients as shown in below picture,

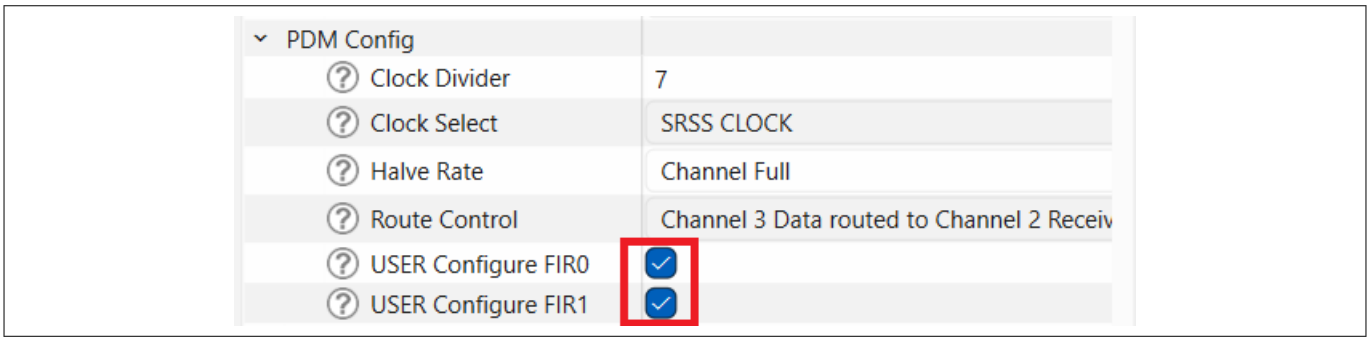


Figure 6 Enable custom FIR filter coefficients

The below pictures show the FIR0 and FIR1 coefficients. Note that the FIR1 coefficients are default and FIR0 are the coefficients that are modified.

FIR0 Config	
FIR0 Coeff0 Data0	5
FIR0 Coeff0 Data1	12
FIR0 Coeff1 Data0	-2
FIR0 Coeff1 Data1	-57
FIR0 Coeff2 Data0	-106
FIR0 Coeff2 Data1	-23
FIR0 Coeff3 Data0	245
FIR0 Coeff3 Data1	466
FIR0 Coeff4 Data0	202
FIR0 Coeff4 Data1	-684
FIR0 Coeff5 Data0	-1509
FIR0 Coeff5 Data1	-1005
FIR0 Coeff6 Data0	1566
FIR0 Coeff6 Data1	5357
FIR0 Coeff7 Data0	8191
FIR0 Coeff7 Data1	8191

Figure 7 FIR0 coefficients

FIR1 Config	
FIR1 Coeff0 Data0	190
FIR1 Coeff0 Data1	258
FIR1 Coeff1 Data0	-106
FIR1 Coeff1 Data1	-87
FIR1 Coeff2 Data0	16
FIR1 Coeff2 Data1	169
FIR1 Coeff2 Data0	-35
FIR1 Coeff2 Data1	-182
FIR1 Coeff3 Data0	-5
FIR1 Coeff3 Data1	226
FIR1 Coeff4 Data0	40
FIR1 Coeff4 Data1	-261
FIR1 Coeff5 Data0	-97
FIR1 Coeff5 Data1	300
FIR1 Coeff6 Data0	173
FIR1 Coeff6 Data1	-336
FIR1 Coeff7 Data0	-276
FIR1 Coeff7 Data1	369
FIR1 Coeff8 Data0	418
FIR1 Coeff8 Data1	-398
FIR1 Coeff9 Data0	-630
FIR1 Coeff9 Data1	422
FIR1 Coeff10 Data0	986
FIR1 Coeff10 Data1	-439
FIR1 Coeff11 Data0	-1764
FIR1 Coeff11 Data1	450
FIR1 Coeff12 Data0	5480
FIR1 Coeff12 Data1	8191

Figure 8 FIR1 coefficients

2.1.3.2 Application flow

The below flowchart shows the application flow for default 16-bit PDM-PCM configuration,

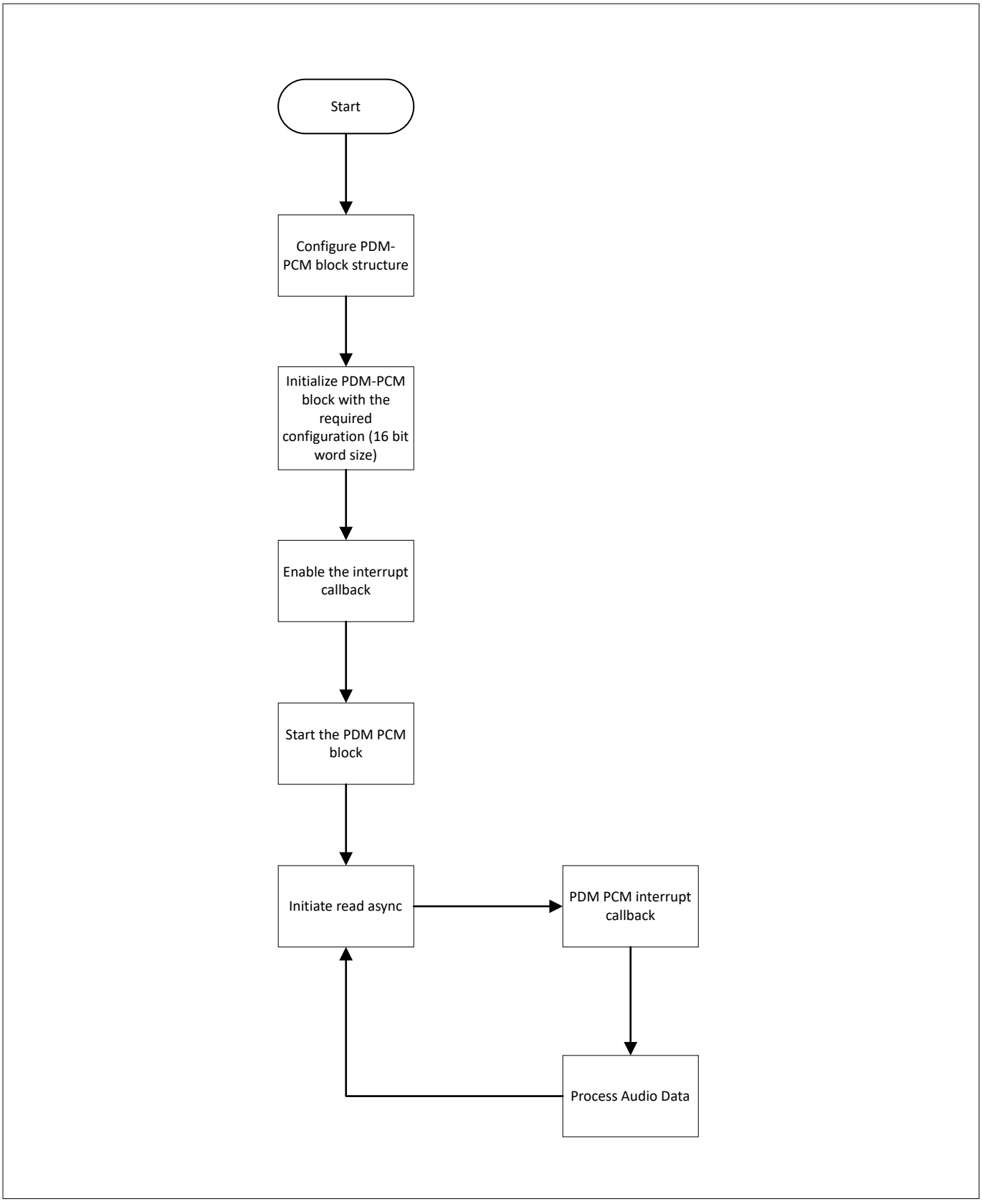


Figure 9 PDM-PCM application flow (16-bit default configuration)

The below flowchart shows the flow with 24-bit PDM-PCM configuration,

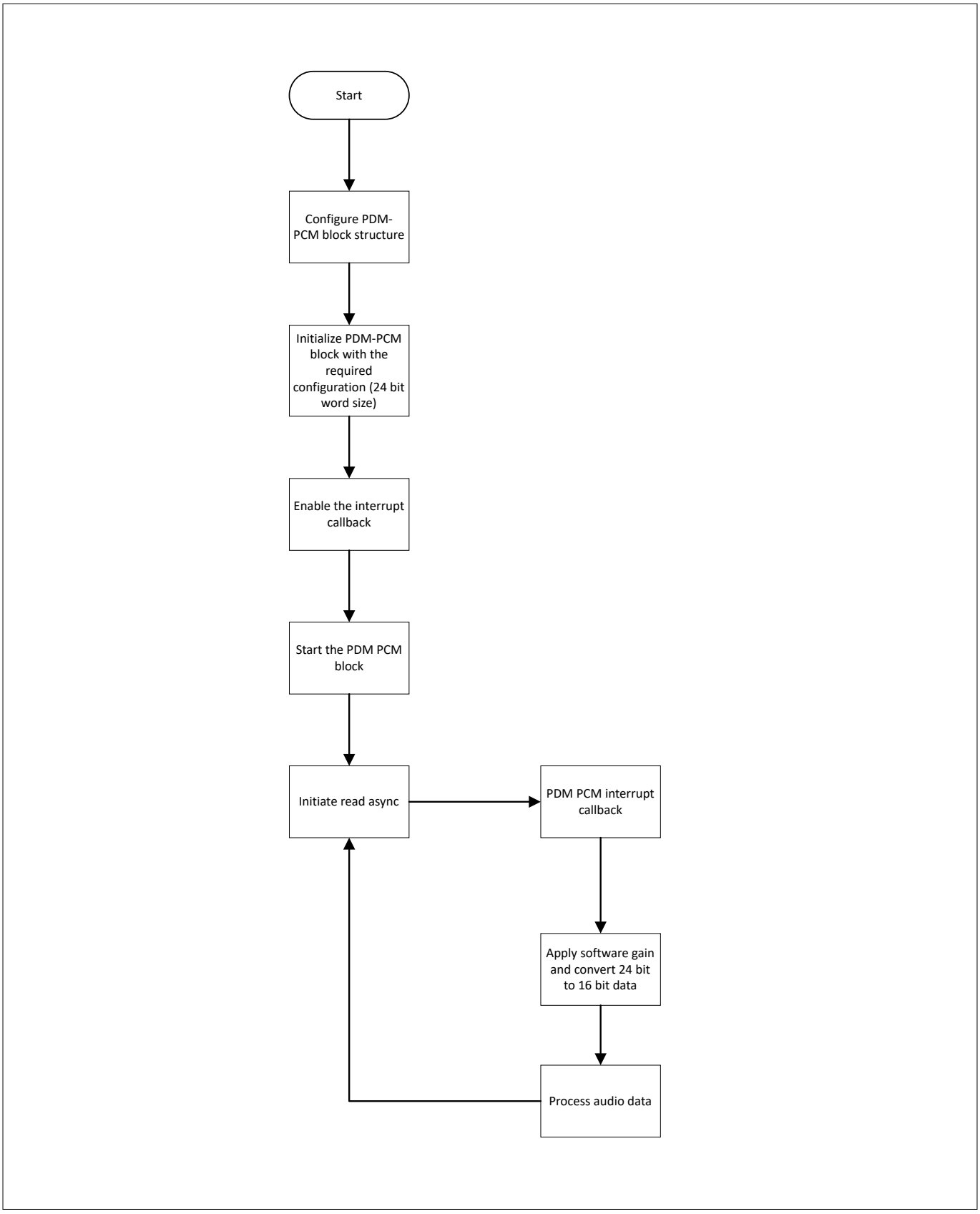


Figure 10 PDM-PCM application flow (24-bit configuration)

With the 24-bit configuration, additionally application has to call Cy_PDM_PCM_ApplyPCM_Gain() API to apply a software gain and convert the data to 16-bit from 24-bit.

***Note**: The audio pipeline in PSOC™ Edge uses 16-bit data, so the conversion is required. The Cy_PDM_PCM_ApplyPCM_Gain() also has an argument that can give the data in 24-bit format itself if required.*

3 DEEPCRAFT™ Audio Enhancement

For processing audio data, Infineon provides DEEPCRAFT™ Audio Enhancement that can process raw audio and provide processed audio for inference, streaming etc.

3.1 Overview

DEEPCRAFT™ Audio Enhancement (AE) enables users to customize the audio processing pipeline running on PSOC™ Edge MCU so that the processed audio can be used for ML inference and other audio related use-cases.

The related Code Example to be used to fully understand Audio Enhancement is named "PSOC™ Edge DEEPCRAFT™ Audio Enhancement Application". https://github.com/Infineon/mtb-example-psoc-edge-ae-application

The AE package consists of following,

- Audio Front End (AFE) Configurator - This runs on the PC and enables user to configure the various audio processing blocks within AFE middleware and also configures the audio pipeline and input/output. It generates a *.c/*.h file which should be flashed to the PSOC™ Edge MCU. Users would need to install the DEEPCRAFT™ Audio Enhancement Tech Pack to access AFE Configurator
- Audio Front End (AFE) Middleware - This is the middleware component that executes on PSOC™ Edge MCU. It has wrapper API and control logic that interfaces between application code and core algorithms
- Audio Voice Core library (audio-voice-core)- This contains the core audio algorithms and is executed on PSOC™ Edge MCU. This is a licensed deliverable from Infineon and supports Arm and LLVM compilers. The licensed version is available via DEEPCRAFT™ Audio pack
- Audio Enhancement (AE) Code Example - This code example (https://github.com/Infineon/mtb-example-psoc-edge-ae-application) is used to demonstrate Audio Enhancement capabilities. Developers can refer to this code example and customize according to their requirement. The README.md and ae_design_guide.md files included in this code example contains details of design, configuration, KPI & other details. There are also other Code Examples such as "PSOC™ Edge Mains Powered Local Voice" that shows how to use the complete audio pipeline with various options and inference engines
- Audio Front End (AFE) Configurator User Guide - Explains how to use AFE configurator
- Audio Enhancement (AE) Quick Start Guide - Quick Start guide on how to setup the AE Code Example
- Audio Enhancement (AE) on PSOC™ Edge E84 MCU - Application note on Audio Enhancement

For more details, refer to https://www.infineon.com/design-resources/embedded-software/deepcraft-edge-ai-solutions/deepcraft-audio-enhancement

Refer to the detailed application note https://www.infineon.com/assets/row/public/documents/30/42/infineon-an240916-deepcraft-audio-enhancement-psoc-edge-applicationnotes-en.pdf

For Quick Start Guide, refer to https://www.infineon.com/assets/row/public/documents/30/44/infineon-deepcraft-audio-enhancement-qsg-usermanual-en.pdf

4 High Performance Core Inference Libraries

4.1 Overview

One of the main use-cases of PSOC™ Edge MCU is to perform ML inference on audio data. PSOC™ Edge MCU has Arm® Cortex® M55 as its high performance core along with Ethos-U55 NPU.

For inference of audio data especially speech data to detect spoken words and commands/intents, PSOC™ Edge MCU can utilize Ethos-U55. The library that is available for detecting spoken words and commands/intents is

- DEEPCRAFT™ Voice Assistant - Inference engine from Infineon (https://www.infineon.com/design-resources/embedded-software/deepcraft-edge-ai-solutions/deepcraft-voice-assistant)

4.2 DEEPCRAFT™ Voice Assistant

4.2.1 Description

DEEPCRAFT™ Voice Assistant is a software solution from Infineon, that enables developers to create natural voice interfaces for their products using AI models that can be deployed to PSOC™ Edge MCU.

Refer to https://www.infineon.com/design-resources/embedded-software/deepcraft-edge-ai-solutions/deepcraft-voice-assistant

The code examples that utilize Voice Assistant are

- https://github.com/Infineon/mtb-example-psoc-edge-voice-assistant-deploy - This is the basic code example which users can use to quickly deploy the models and see the output on UART
- https://github.com/Infineon/mtb-example-psoc-edge-mains-powered-local-voice - This code example demonstrates the complete audio pipeline with tunable Audio Enhancement, USB audio capability and use-case with switchable graphics. The default use-case is music player. Users can customize this CE for any of the use-case. Refer to the https://github.com/Infineon/mtb-example-psoc-edge-mains-powered-local-voice/blob/master/docs/local_voice_design_guide.md for more details

4.2.2 Training Voice Assistant Models

The training of Voice Assistant models is done in cloud and then the trained models are deployed to PSOC™ Edge MCU.

The Voice Assistant models utilize Arm® Cortex® M55 + Ethos-U55 on PSOC™ Edge MCU.

On a high level, the following are the steps required to train a model for Voice Assistant,

Login to https://deepcraft-voice-assistant.infineon.com/welcome

After logging in, select **Design New Project** for a new project

or,

if you have an existing *.vaproj* file, select **Import Project**



Figure 11 Voice Assistant New/Import Project

For adding a wakeword, like "Ok Infineon", select as shown below,

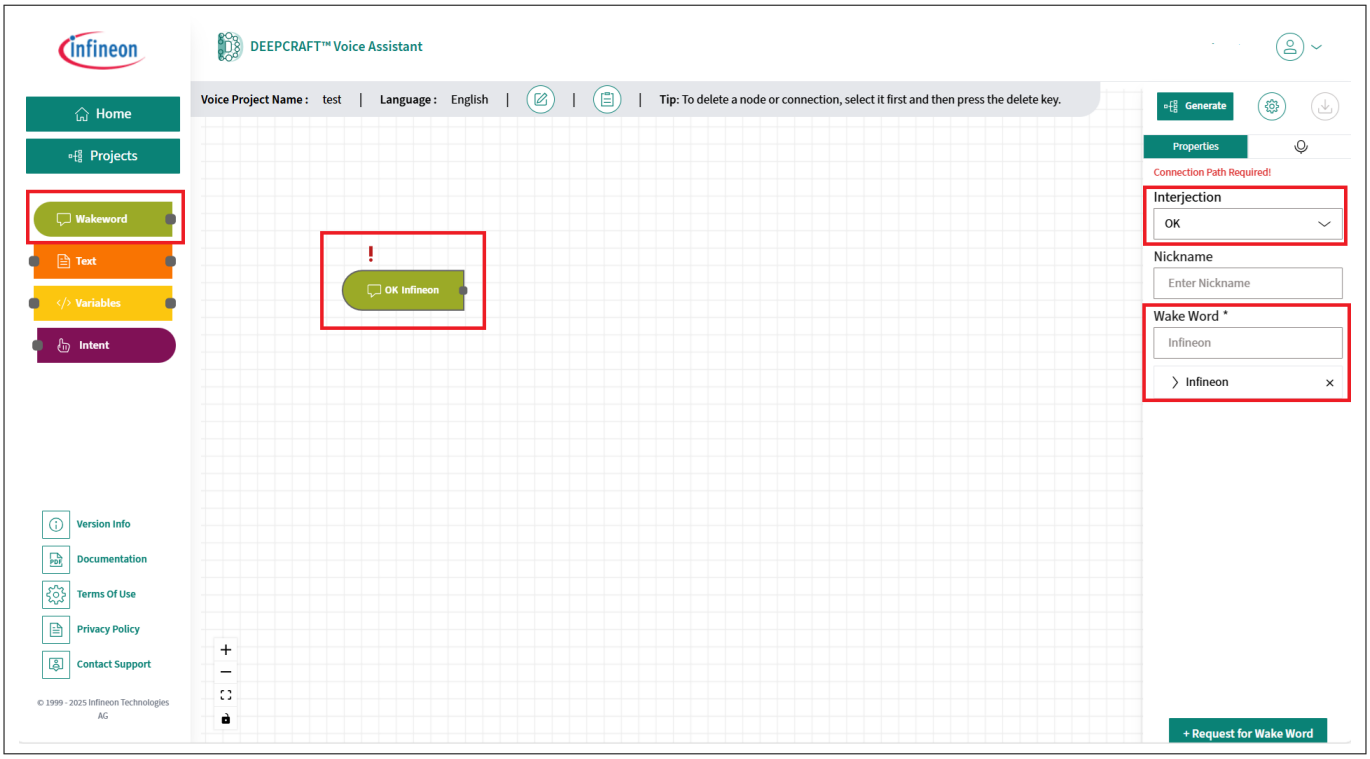


Figure 12 Adding Wakeword

To add the commands, select the **Text** options. To add numbers, select **Variables**. Ensure to add the required details in the **Properties** tab on the right.

***Note**: In the image below, there are **Optional** and **Required** words in the command flow. It shows "can you play the song" where "can you" is optional word*

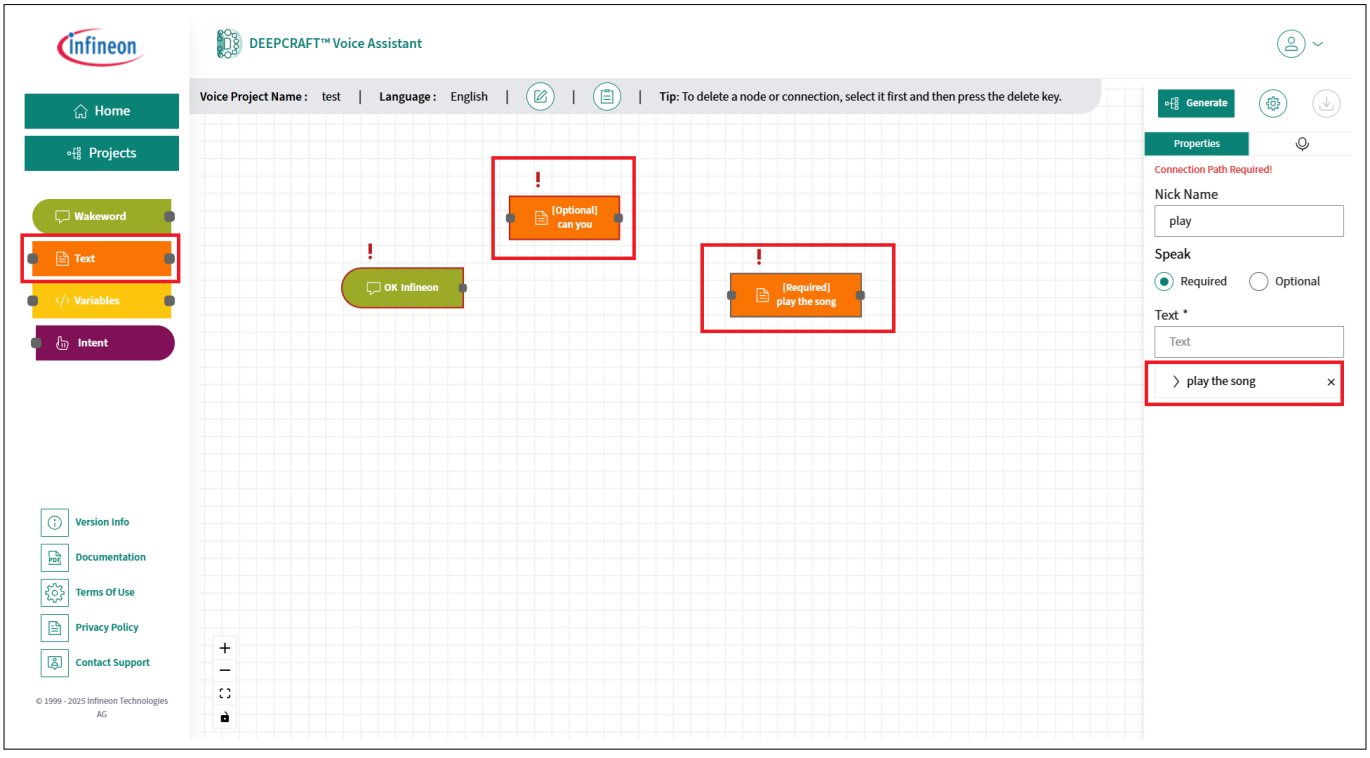


Figure 13 Adding commands

Each command flow should be mapped to an intent. Choose the **Intent** option and add the intent label. Ensure to connect all the nodes as shown in below picture,

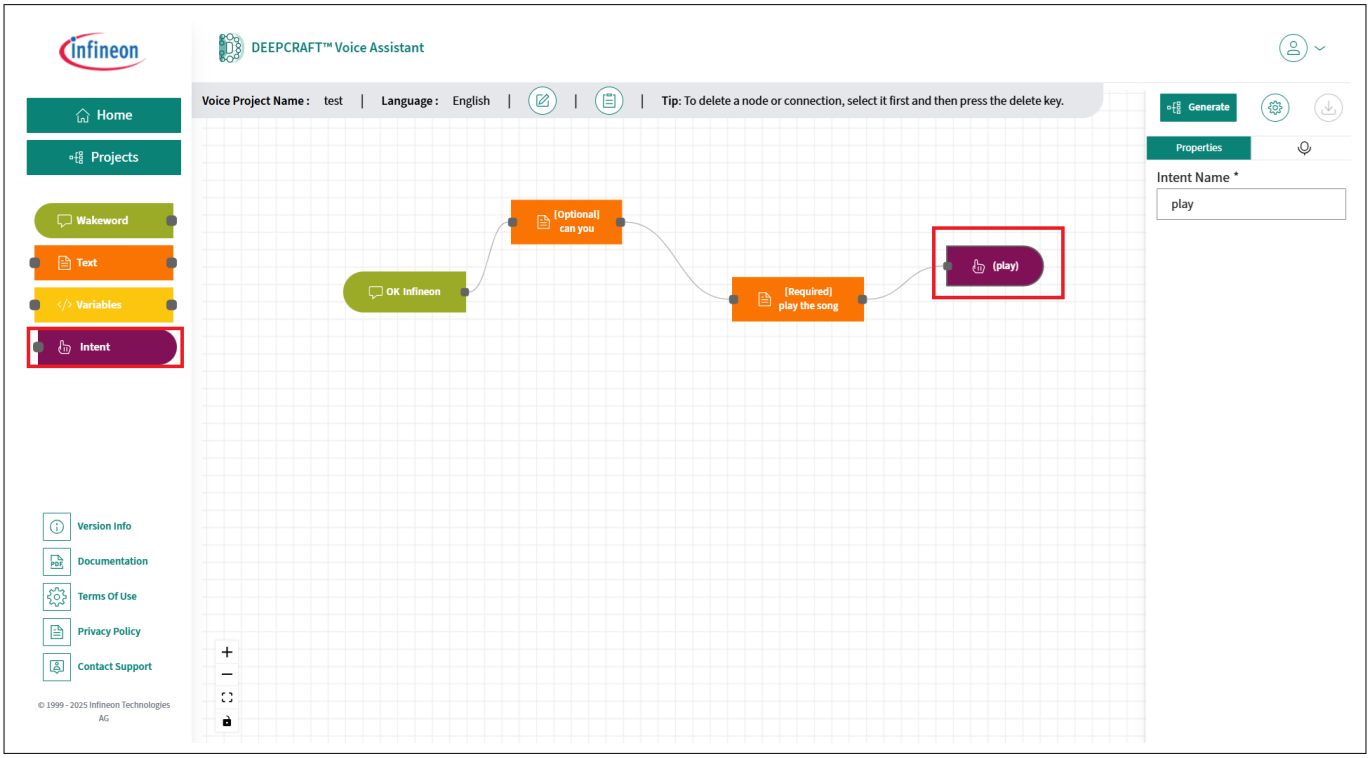


Figure 14 Intent mapping

After adding all the required commands and intents, the model graph would look similar to

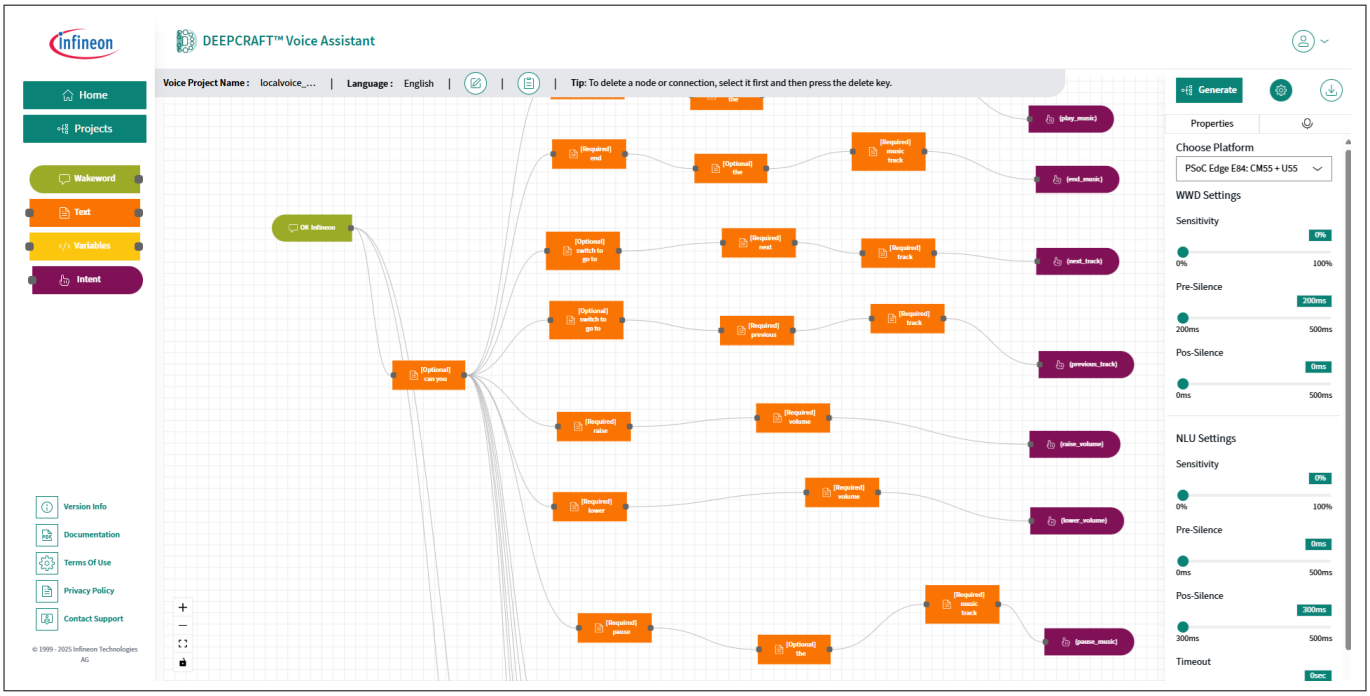


Figure 15 Model graph

Select Generate to generate the model.

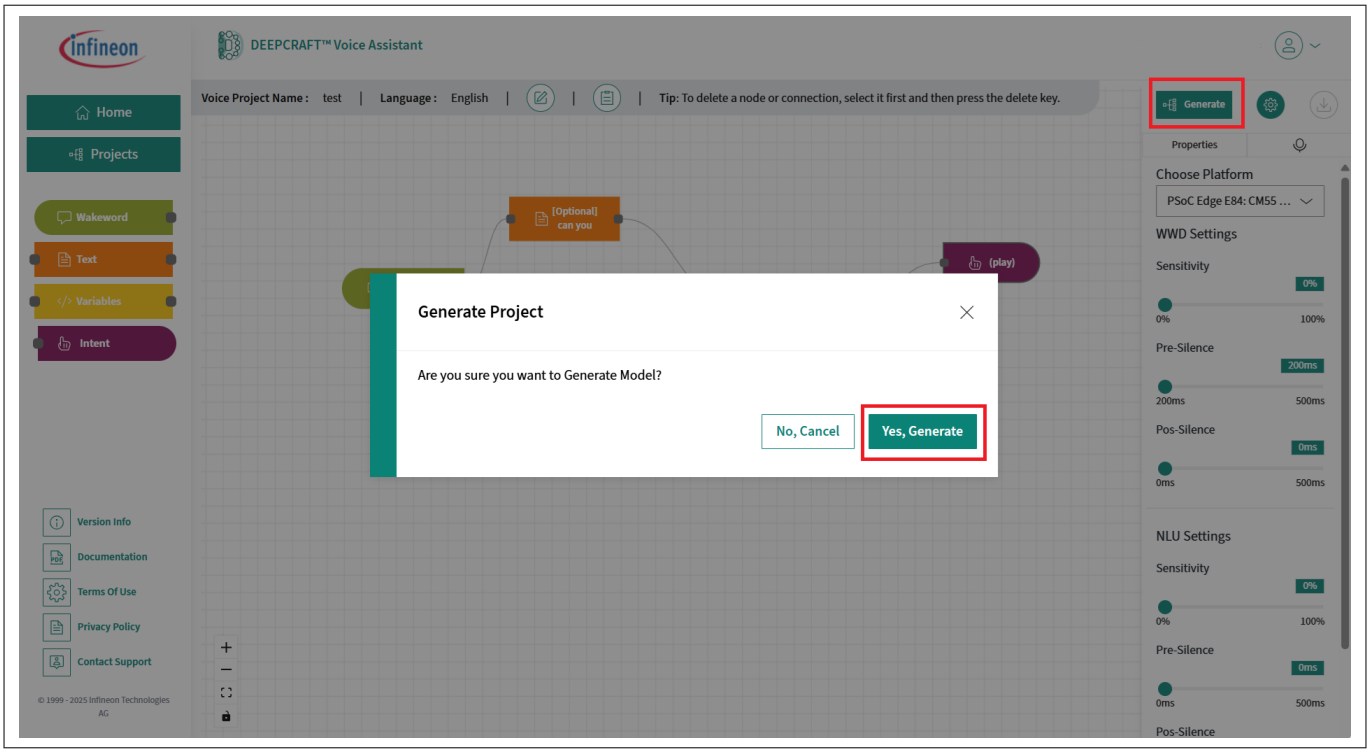


Figure 16 Generate model

Users can further refer to https://www.infineon.com/assets/row/public/documents/30/44/infineon-deepcraft-voice-assistant-user-guide-usermanual-en.pdf for understanding the cloud interface and steps to train the model.

4.2.3 Integration to code example

Once the training is done, the Voice Assistant models can be integrated to the code example.

The first step is to download the files from the Voice Assistant cloud project,

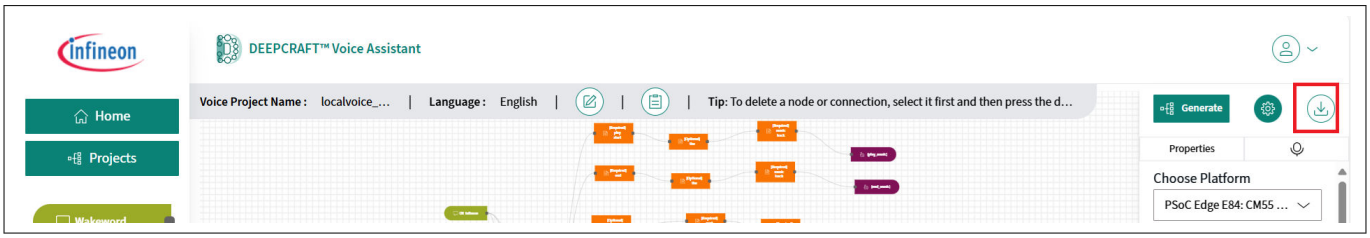


Figure 17 Download model files

Users can now use either of the two code examples to integrate the model to the code,

- https://github.com/Infineon/mtb-example-psoc-edge-voice-assistant-deploy - For quickly evaluating the deployed model and check if it can detect the spoken wakeword and commands. In this code example, the detected commands/intents are printed on UART. Refer to README.md and design_and_implementation.md of this code example for more details
- https://github.com/Infineon/mtb-example-psoc-edge-mains-powered-local-voice/ - For actual use-cases that require the full audio pipeline with customizable Audio Enhancement components, USB Audio capture and feed for debugging, switchable code for use-cases etc. Refer to README.md and local_voice_design_guide.md available in this code example for details

5 Audio Software Codec

5.1 Software Codecs Overview

Software codecs, in the context of audio applications, perform compression and decompression of audio streams to reduce bandwidth requirements and storage footprint while meeting target audio quality and latency. A codec consumes audio data in one format (typically linear PCM) and produces an encoded bitstream; a matching decoder reconstructs PCM from that bit-stream. In most applications, codecs are used in pairs: encoder on the originating device and decoder on the receiving device.

In MCU, the software codecs are designed to run on the CM55 (High Performance) CPU as well as CM33 (Low Power) CPU. Software codecs are enabled through the software codec middleware component. The middleware abstracts codec-specific details behind a common API, enabling applications to select a codec and configure parameters such as sampling rate, channels, bit width, frame duration, and bitrate. For details on the software codec middleware, refer to the audio-sw-codecs GitHub repository.

5.2 Types of Software Codecs

The MCU supports multiple software codecs on its dual-CPU architecture. The details of the codecs available on each core, their capabilities, and implementation considerations for audio applications are shown in the below tables for CM55 CPU and CM33 CPU.

Table 2 Audio Software Codec support on CM55 CPU

Codec	Decode support	Encode support
SBC	Yes	Yes
OPUS*	Yes	Yes
LC3*	Yes	Yes
MP3*	Yes	No
AAC*	Yes	No

Table 3 Audio Software Codec support on CM33 CPU

Codec	Decode support	Encode support
SBC	Yes	Yes
OPUS*	No	No
LC3*	Yes	Yes
MP3*	No	No
AAC*	No	No

***Note**: Codecs marked with an * require an additional technology pack. Please use ModusToolbox™ Setup to install the ModusToolbox™ Audio SW Codecs Tech Pack.*

5.2.1 SBC codec

Low Complexity Subband Coding (SBC) is designed for Bluetooth® audio transport. SBC uses subband filtering, adaptive block PCM quantizers, and adaptive bit allocation to achieve high perceived quality at medium bitrates with modest computational complexity.

SBC is the mandatory codec for Bluetooth® A2DP (Advanced Audio Distribution Profile). Additionally, the same codec with parameter modifications can be used as mSBC (modified SBC) for Bluetooth® HFP (Hands-Free Profile) voice applications.

Codec Details:

- Typical sampling rates: 16 kHz, 32 kHz, 44.1 kHz, 48 kHz.
- Bit width: 16-bit PCM.
- Channel modes: mono, dual channel, stereo, joint stereo.
- Block lengths: 4, 8, 12, 16.
- Subbands: 4 or 8.
- Allocation methods: loudness or SNR.
- Bitpool drives bitrate and quality; common ranges for joint stereo at 44.1/48 kHz are roughly 30–53 for medium to higher quality. At 16 kHz, lower bitpools are typical.
- Frame duration depends on subbands and blocks; practical end-to-end audio latency in A2DP/HFP applications is dominated by buffering and transport, not the encoder's algorithmic delay.
- Compression ratio: 4:1 to 8:1 typical.

Standards and documentation: See Bluetooth® SIG specifications for SBC implementation details and A2DP/HFP profile requirements.

5.2.2 Opus codec

Opus is a versatile, real-time audio codec standardized by IETF (Internet Engineering Task Force). Opus codec consists of a Linear Prediction (LP) layer and a Modified Discrete Cosine Transform (MDCT) layer. It is a real-time interactive codec for speech and music transmission over a network. It is also used for streaming and storage of audio. OPUS codec is used for wide range of applications.

The LP layer is used for low frequencies and MDCT layer is used for higher frequencies. This combination provides better quality of audio encoding. LP layer is based on SILK codec and MDCT later is based on Constrained-Energy Lapped Transform (CELT) codec. Opus automatically and seamlessly switches or blends these modes to optimize quality, bitrate, and latency. Opus can dynamically switch between SILK, CELT, or hybrid mode that combines both techniques.

Codec Details:

- Bit width: 16-bit PCM.
- Sampling rate: 16 kHz, 32 kHz, 48 kHz (44.1 kHz is not supported by the middleware).
- Channels: mono or stereo.
- Frame duration: 10 ms and 20 ms.
- Bitrate: CBR only (VBR is not supported by the middleware).
- Compression ratio: Up to 12:1 for speech, 6:1 for music.

Standards and documentation: See the IETF site for RFCs and the Opus community site for technical documentation of the Opus codec.

5.2.3 LC3 codec

Low Complexity Communication Codec (LC3) is a next-generation audio codec designed primarily for Bluetooth® LE Audio applications. LC3 provides superior audio quality at lower bitrates compared to traditional codecs, making it ideal for power-constrained devices and bandwidth-limited applications.

LC3 uses advanced signal processing techniques including Modified Discrete Cosine Transform (MDCT), spectral noise shaping, and temporal noise shaping to achieve high compression efficiency while maintaining low computational complexity and minimal algorithmic delay.

Codec Details:

- Bit width: 16-bit PCM input.
- Sampling rates: 8 kHz, 16 kHz, 24 kHz, 32 kHz, 44.1 kHz, 48 kHz.
- Channels: mono or stereo.
- Frame duration: 7.5 ms and 10 ms.
- Compression ratio: Up to 10:1 typical.

Standards and documentation: See Bluetooth® SIG specifications for LC3 implementation details and LE Audio profile requirements. LC3 is standardized as part of the Bluetooth® LE Audio specification.

5.2.4 MP3 codec

MPEG-1 Audio Layer III (MP3) is a widely adopted lossy audio compression format that revolutionized digital audio distribution. MP3 uses psychoacoustic models to remove perceptually irrelevant information from audio signals, achieving significant compression ratios while maintaining acceptable audio quality.

The codec employs a hybrid filter bank with 32 subbands using polyphase filtering, followed by Modified Discrete Cosine Transform (MDCT) for frequency domain processing. Bit allocation is performed using psychoacoustic models that mask quantization noise below the threshold of human hearing.

Codec Details:

- Bit width: 16-bit PCM.
- Sampling rates: 16 kHz, 32 kHz, 44.1 kHz, 48 kHz.
- Channels: mono, stereo, joint stereo, dual channel.
- Supported operations: Decoding (MP3 encoding is currently not supported by the middleware).
- Bitrates: 32 kbps to 320 kbps (CBR), VBR also supported.
- Frame size: 1152 samples per frame.
- Compression ratio: Up to 12:1 typical for CD-quality audio.

Standards and documentation: See ISO/IEC 11172-3 (MPEG-1 Audio) and ISO/IEC 13818-3 (MPEG-2 Audio) specifications for MP3 implementation details and technical requirements.

5.2.5 AAC codec

High-Efficiency Advanced Audio Coding (HE-AAC) is a lossy audio compression standard designed as an enhanced version of AAC. HE-AAC combines AAC Low Complexity (AAC-LC) with Spectral Band Replication (SBR) and sometimes Parametric Stereo (PS) to achieve superior quality at lower bitrates.

HE-AAC uses a two-layer approach: the core AAC-LC encoder handles lower frequencies, while SBR reconstructs higher frequencies from lower frequency information and additional side information. This technique allows for efficient encoding of both speech and music content.

Codec Details:

- Bit width: 16-bit PCM.
- Sampling rates: 16 kHz, 32 kHz, 44.1 kHz, 48 kHz.
- Channels: mono, stereo.
- Supported input formats: ADTS, ADIF.
- Supported operations: Decoding (HE-AAC encoding is currently not supported by the middleware).
- Compression ratio: Up to 20:1 for speech, 10:1 for music.

Standards and documentation: See ISO/IEC 14496-3 (MPEG-4 Audio) specifications for AAC implementation details. HE-AAC is defined in ISO/IEC 14496-3:2009/Amd.2:2010.

5.3 Code Flow

The audio software codec library provides a unified interface for encoding and decoding operations across multiple codec types. The following sections detail the typical workflow for using the codecs in applications.

Project Configuration

Add the required codec components to your project's Makefile:

```
# Enable required codecs
COMPONENTS+=SBC_CODEC OPUS_CODEC LC3_CODEC MP3_CODEC HEAAC_CODEC

# Define codec-specific macros
ifneq (,$(findstring SBC_CODEC,$(COMPONENTS)))
SBC_CODEC_DEFINES=ENABLE_SBC_DECODE ENABLE_SBC_ENCODE
endif

ifneq (,$(findstring OPUS_CODEC,$(COMPONENTS)))
OPUS_CODEC_DEFINES=HDR_WRITE ENABLE_OPUS_ENCODE OPT_HDR ENABLE_OPUS_DECODE
endif

ifneq (,$(findstring LC3_CODEC,$(COMPONENTS)))
LC3_CODEC_DEFINES=ENABLE_LC3_DECODE ENABLE_LC3_ENCODE
endif

ifneq (,$(findstring MP3_CODEC,$(COMPONENTS)))
MP3_CODEC_DEFINES=SUPPORT_MP3_PARSING ENABLE_MP3_DECODE
endif

ifneq (,$(findstring HEAAC_CODEC,$(COMPONENTS)))
HEAAC_CODEC_DEFINES=ENABLE_HEAAC_DECODE
endif

DEFINES+=$(SBC_CODEC_DEFINES) $(OPUS_CODEC_DEFINES) $(LC3_CODEC_DEFINES) $(MP3_CODEC_DEFINES) $
(HEAAC_CODEC_DEFINES)
```

API Usage

The high-level code flow for encoding and decoding operations are shown in Figure 18 and Figure 19.

5 Audio Software Codec

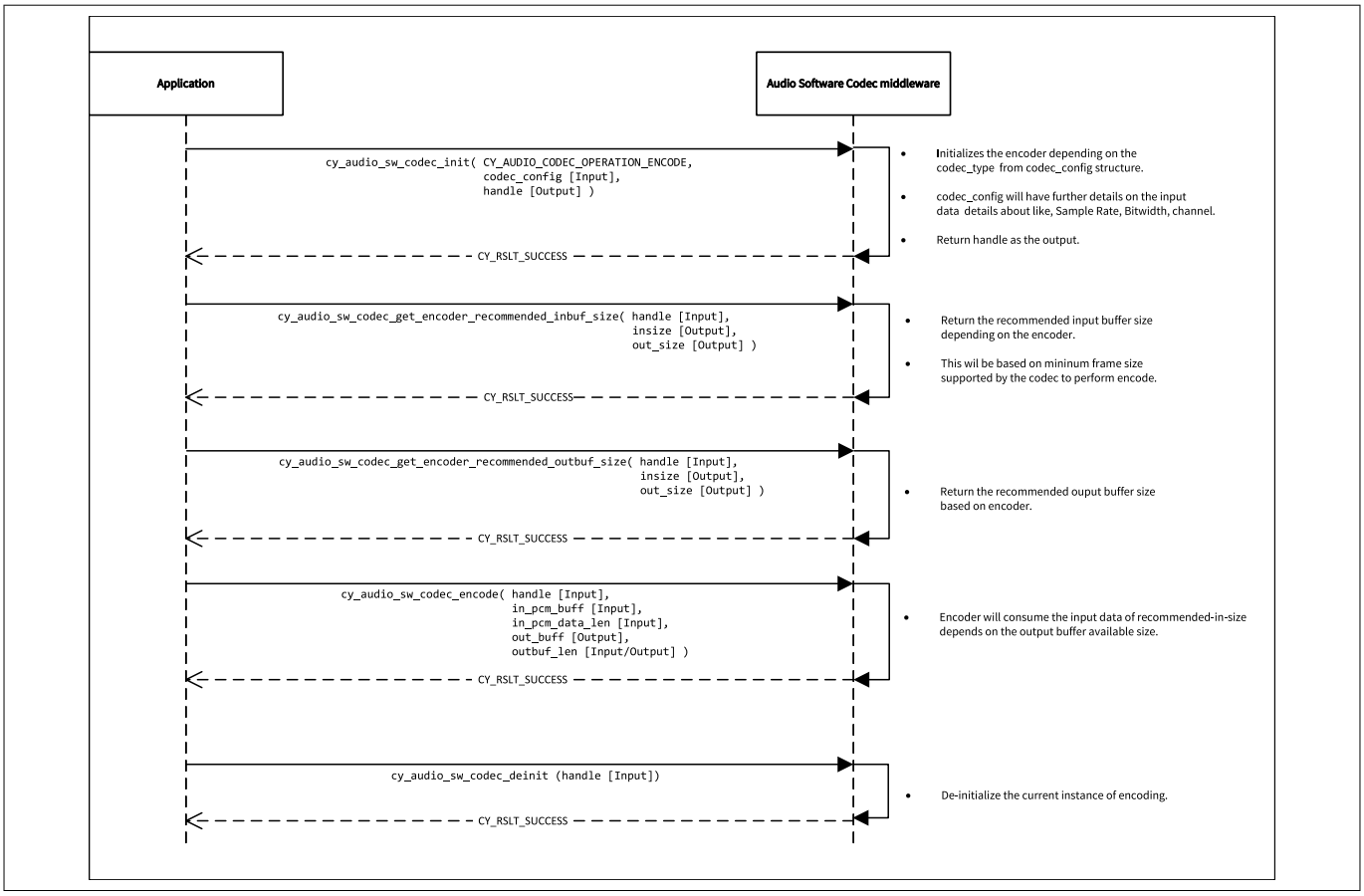


Figure 18 API sequence for encoding using audio software codecs

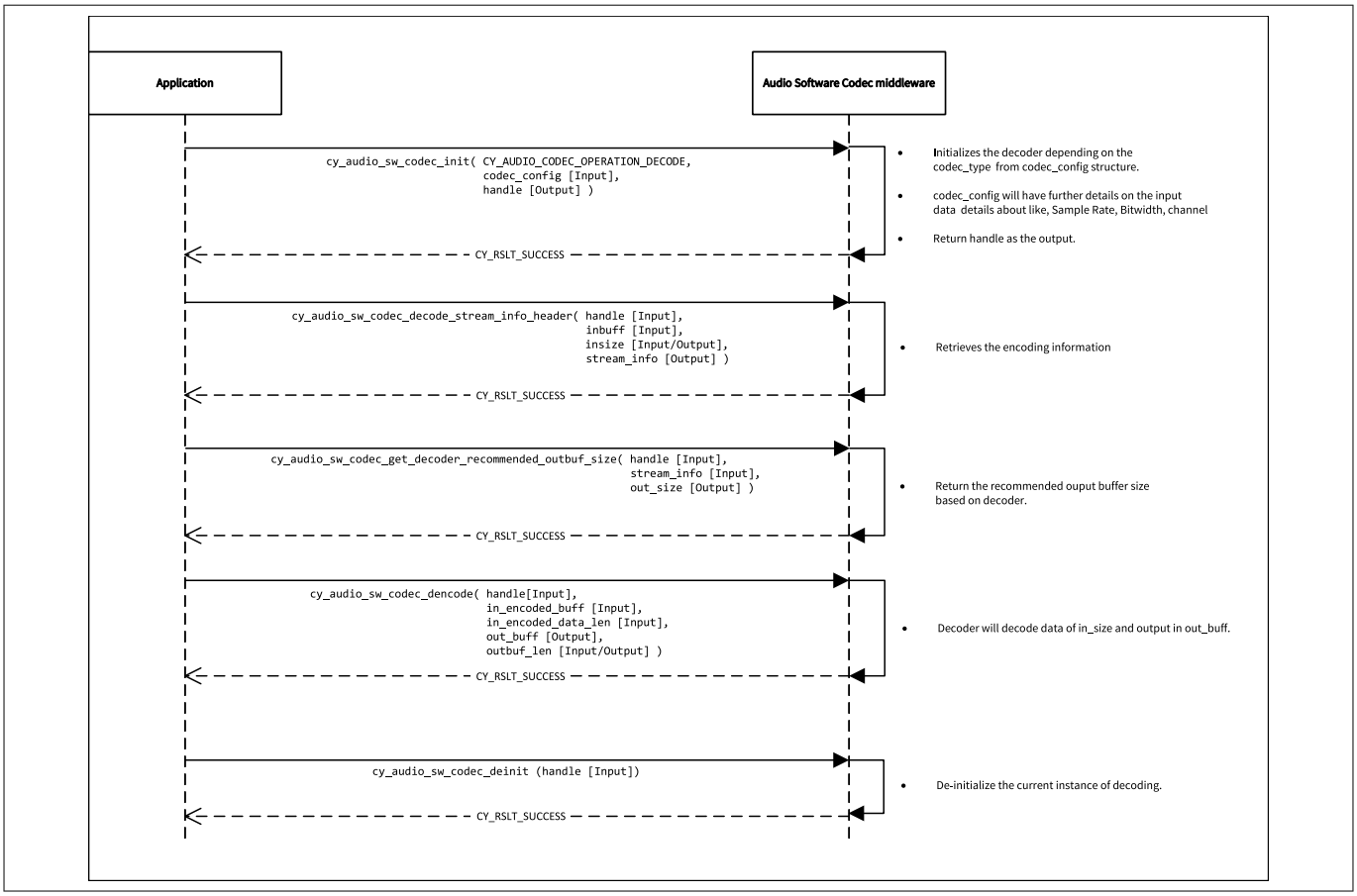


Figure 19 API sequence for decoding using audio software codecs

Performance Considerations

- Use appropriate frame sizes for your application's latency requirements
- Place critical codec API's text and data segments in faster memories for enhanced performance
- Using Helium extension of CM55 CPU (using supported compiler toolchains) can improve performance significantly
- Consider buffer management for real-time applications
- Use RTOS priorities appropriately for time-critical codec operations
- Profile memory usage, especially for multiple concurrent codec instances

6 Bluetooth®/Bluetooth® Low Energy (BT/BLE) for Audio & Voice

6.1 Introduction to Bluetooth® for Audio & Voice Applications

6.1.1 System architecture

A typical architecture used in Bluetooth® based use-cases involves the usage of two entities – Bluetooth® Host and Controller.

- The Host is an entity that runs the end application, and a portion of the Bluetooth® stack called the Host Stack (in this case, the Host Stack is referred to as the BT Stack). The Host is typically the main system processor of the system, and it is mostly responsible for more complex implementations but with less stringent timing requirements.
- The Bluetooth® Controller is a separate entity that controls the low-level Bluetooth® radio along with link management. Since the controller deals with hard real-time requirements and contact with the physical layer, it is good to separate this part from the host.

The Host Controller Interface (HCI) is the interface that enables communication between the Host and the Controller. The HCI indicates the logical interface between them but is not a physical component as such. HCI can be implemented in a number of different ways in terms of the underlying physical transport, but the logical or functional interface is always the same.

The Bluetooth® specifications specify the Host and Controller as two logical entities only and do not mandate any implementation details. But in practice, two common architectures are prevalent:

1. The Host and Controller as two separate hardware devices with a physical connection in between them.
2. The Host and Controller reside on the same hardware device with just a logical separation.

In the scope of this document for applications, the type 1 mentioned above is the architecture being used. In this case, the Host is the MCU and the Bluetooth® controller could be an AIROC™ Wi-Fi and Bluetooth® combo device like CYWxxxxx (that is, CYW5551x, CYW4343x, CYW43012, CYW4373x). The HCI used in this case is a 4-wire UART interface with flow control. This configuration is shown in Figure 20.

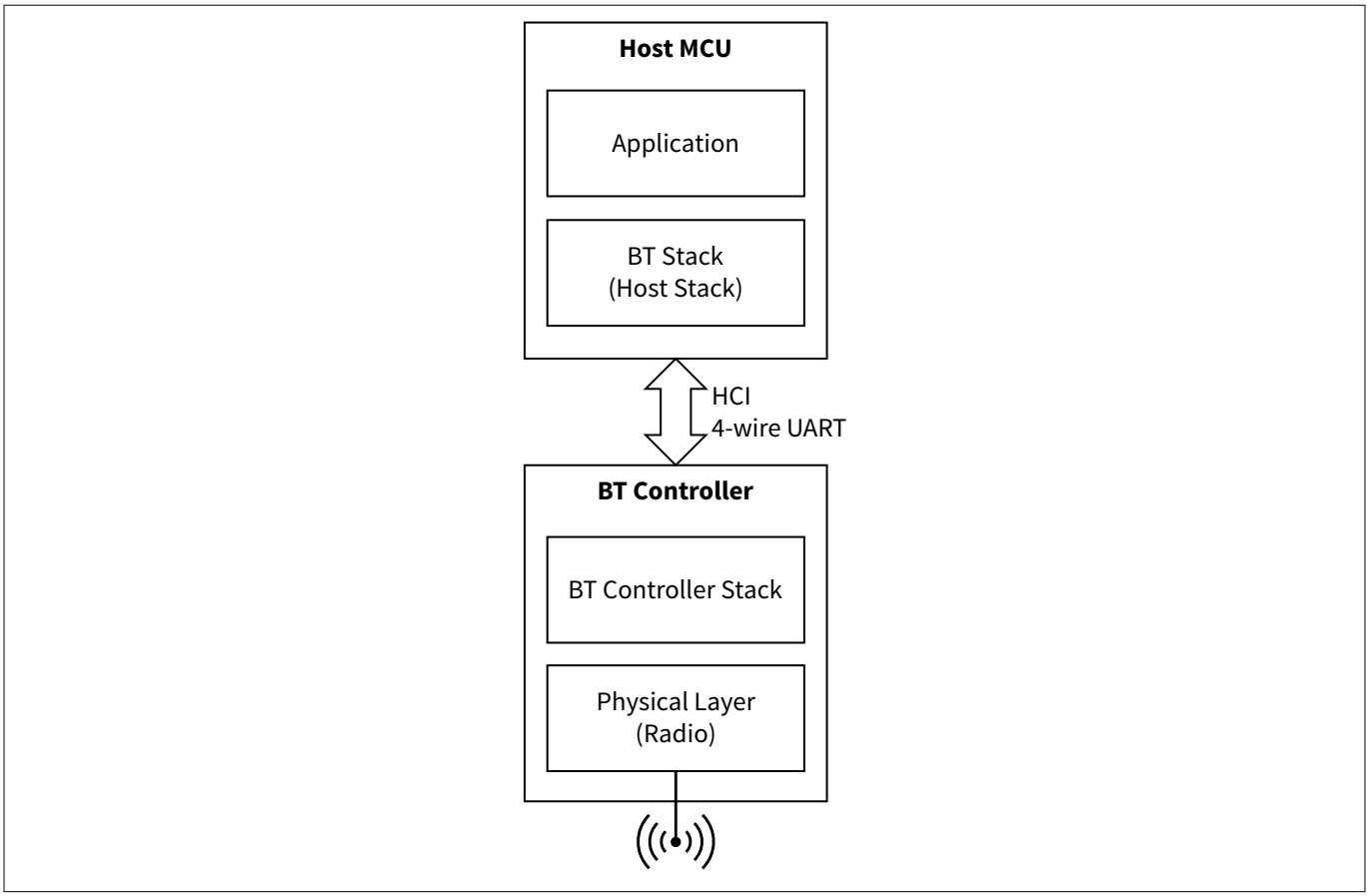


Figure 20 Typical Bluetooth® system architecture

6.1.2 Bluetooth® stack

Figure 21 illustrates the various layers present in the Bluetooth® Protocol Stack for the Bluetooth® Classic and Bluetooth® Low Energy technologies.

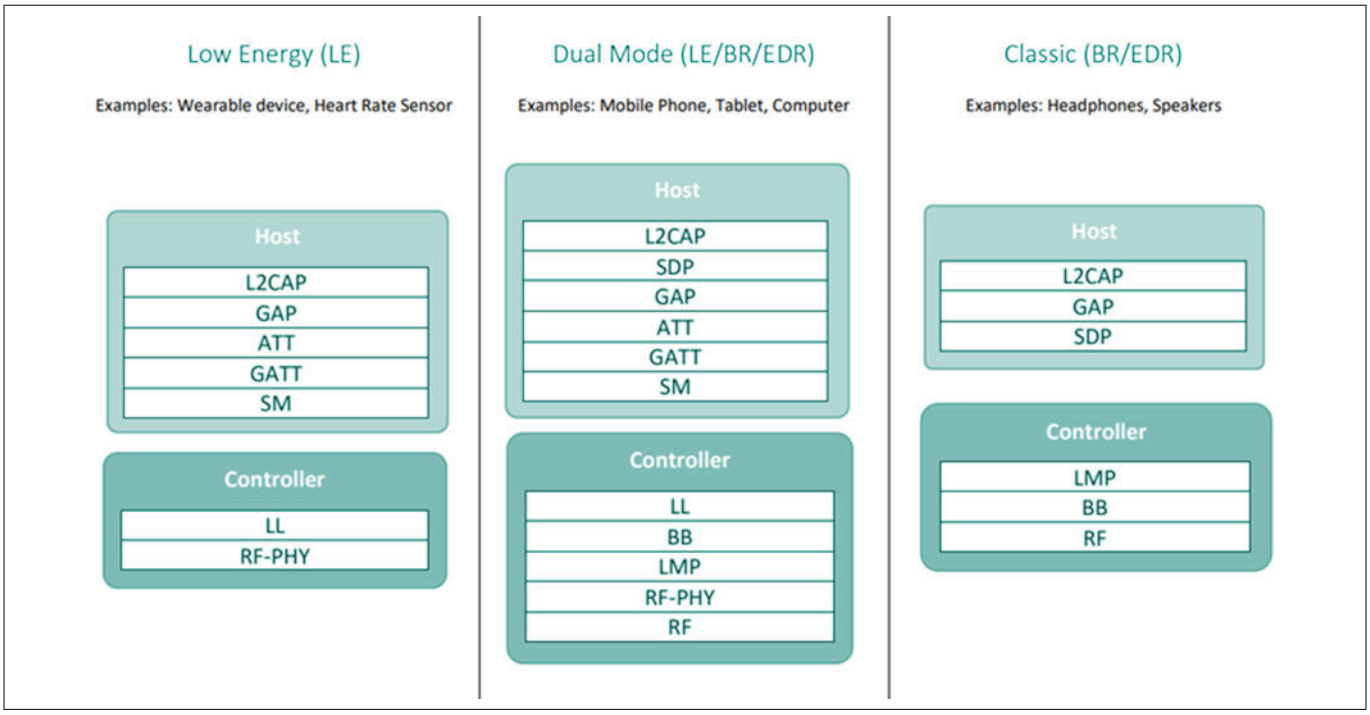


Figure 21 Bluetooth® Protocol Stack

The stack in the first column depicts the Bluetooth® Protocol Stack for the Bluetooth® Low Energy applications. In this case none of the Bluetooth® Classic Profiles are available.

Similarly, the stack in the last column depicts the Bluetooth® Protocol Stack for the Bluetooth® Classic applications. In this case none of the BLE Profiles are available.

The stack shown in the middle illustrates the Bluetooth® Protocol Stack for both Bluetooth® Low Energy and Bluetooth® Classic. This mode of having both Bluetooth® Low Energy and Bluetooth® Classic (BR/EDR) is known as Dual Mode Bluetooth®.

This Dual Mode Bluetooth® stack is what is desired and used in the context of this document. The layers above the HCI run on the Host device (MCU) and the layers below the HCI run on the BT controller device (AIROC™ Wi-Fi and Bluetooth® combo device - CYWxxxxx).

6.1.3 Bluetooth® profiles for Audio & Voice Applications

The Bluetooth® Profiles for Audio & Voice Applications can be categorized as:

1. Bluetooth® Classic Profiles
2. Bluetooth® Low Energy Profiles

The details of the profiles under these categories are as follows.

1. Bluetooth® Classic Profiles

The following Bluetooth® Classic Profiles are targeted for Bluetooth®audio and voice applications:

1. Advanced Audio Distribution Profile (A2DP)
2. A/V Remote Control Profile (AVRCP)
3. Hands-Free Profile (HFP)

The details of these profiles are given below.

1.1. Advanced Audio Distribution Profile (A2DP)

Advanced Audio Distribution Profile (A2DP) describes how stereo-quality audio can be streamed from a media source to a sink.

The source is typically a device that generates or stores audio content, like a smartphone, tablet, laptop, or a digital media player and is responsible for sending/transmitting the audio signal.

The sink is the device that receives and decodes the transmitted audio signal. This could be headphones, speakers, a car stereo system, or any device equipped with Bluetooth® audio reception capabilities.

The profile defines two roles of an audio device: source and sink:

- A2DP Source: A device is the source when it acts as a source of a digital audio stream that is delivered to the SINK
- A2DP Sink: A device is the sink when it acts as a sink of a digital audio stream delivered from the SOURCE

A2DP defines the protocols and procedures that realize the distribution of high-quality audio content of high-quality in mono or stereo on ACL channels. The term “advanced audio,” therefore, should be distinguished from “Bluetooth® audio,” which indicates the distribution of narrow-band voice on SCO channels as defined in the baseband specification. For more details on the A2DP Bluetooth® Specifications, please refer to the link: https://www.bluetooth.com/specifications/specs/advanced-audio-distribution-profile-1-4/.

1.2. A/V Remote Control Profile (AVRCP)

A/V Remote Control Profile (AVRCP) provides an interface for controlling different Bluetooth® media devices such as TVs, music players and headphones remotely from a Bluetooth® controller device. Typical supported functions are play, pause, stop, next, and previous. It is usually coupled with A2DP so that, for example, wireless headphones use A2DP to stream the music data and AVRCP for the control data to control the media playback.

There are two roles defined for devices using AVRCP:

- Controller (CT): device that initiates a transaction by sending a command frame to a target. Examples: personal computer, PDA, mobile phone, remote controller or an AV device (such as an in car system, headphone, player/recorder, timer, tuner, monitor etc.)
- Target (TG): device that receives a command frame and accordingly generates a response frame. Examples: audio player/recorder, video player/recorder, TV, tuner, amplifier or headphone

For more details on the AVRCP Bluetooth® Specifications, please refer to the link: https://www.bluetooth.com/specifications/specs/a-v-remote-control-profile-1-6-2/.

1.3. Hands-Free Profile (HFP)

The Hands-Free Profile (HFP) specification defines a set of functions such that a Mobile Phone can be used in conjunction with a Hands-Free device (e.g., installed in the car or represented by a wearable device such as a headset, or smartwatch), a Bluetooth® Link providing a wireless means for both remote control of the Mobile Phone by the Hands-Free device and voice connections between the Mobile Phone and the Hands-Free device.

The following roles are defined for this profile:

- Audio Gateway (AG): This is the device that is the gateway of the audio, both for input and output. Typical devices acting as Audio Gateways are cellular phones
- Hands-Free unit (HF): This is the device acting as the Audio Gateway’s remote audio input and output mechanism. It also provides some remote control means like answering, rejecting calls, initiating calls, etc. The HFP HF is also referred to as the HFP Unit

For more details on the HFP Bluetooth® Specifications, please refer to the link: https://www.bluetooth.com/specifications/specs/hands-free-profile/.

2. Bluetooth® Low Energy Profiles

Applications using Bluetooth® Low Energy (BLE) services require various profiles for normal functioning like GAP, ATT, GATT, etc. GAP and GATT layers abstract all the underlying BLE layers. GAP provides a framework for controlling BLE devices and GATT provides a framework for managing data in BLE devices. In the scope of this document, only the GATT and Bluetooth® Low Energy Audio Profiles are briefly described in the context of audio and voice applications and the other profiles are not covered although necessary for operation. For more details on the getting started with BLE on MCU, please refer to the Getting started with *PSOC™ MCU and AIROC™connectivity devices* application note.

2.1. Bluetooth® Low Energy GATT

GATT is an acronym for the Generic ATTribute Profile, and it defines the way that two Bluetooth® Low Energy devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and

related data in a simple lookup table using 16-bit IDs for each entry in the table. It relies on a client-server architecture:

- Servers: These are devices that contain data and make it available to the clients. In the BLE context, these servers are usually peripherals such as a heart rate monitor
- Clients: These are the devices that connect to the server to access or modify the server's data. Typically, these are computing devices like a smartphone or a tablet that use applications to interact with the server's data

In the context of Bluetooth®audio and voice applications, BLE GATT can be used in use-cases that transmit compressed, low-bitrate voice frames between the MCU and a smartphone acting as the gateway to the cloud. These implementations typically define custom GATT services and characteristics for audio payloads, control signaling, and status, encapsulating a proprietary framing protocol over ATT. While GATT provides a secure, reliable, and power-efficient means for transporting voice data, it is best suited for narrowband, latency-sensitive streams rather than high-fidelity audio. In contrast, standardized LE Audio transports continuous audio over isochronous channels and uses GATT primarily for discovery and control; therefore, GATT-based voice transport should be carefully engineered around codec choice, packetization, flow control, and smartphone platform constraints.

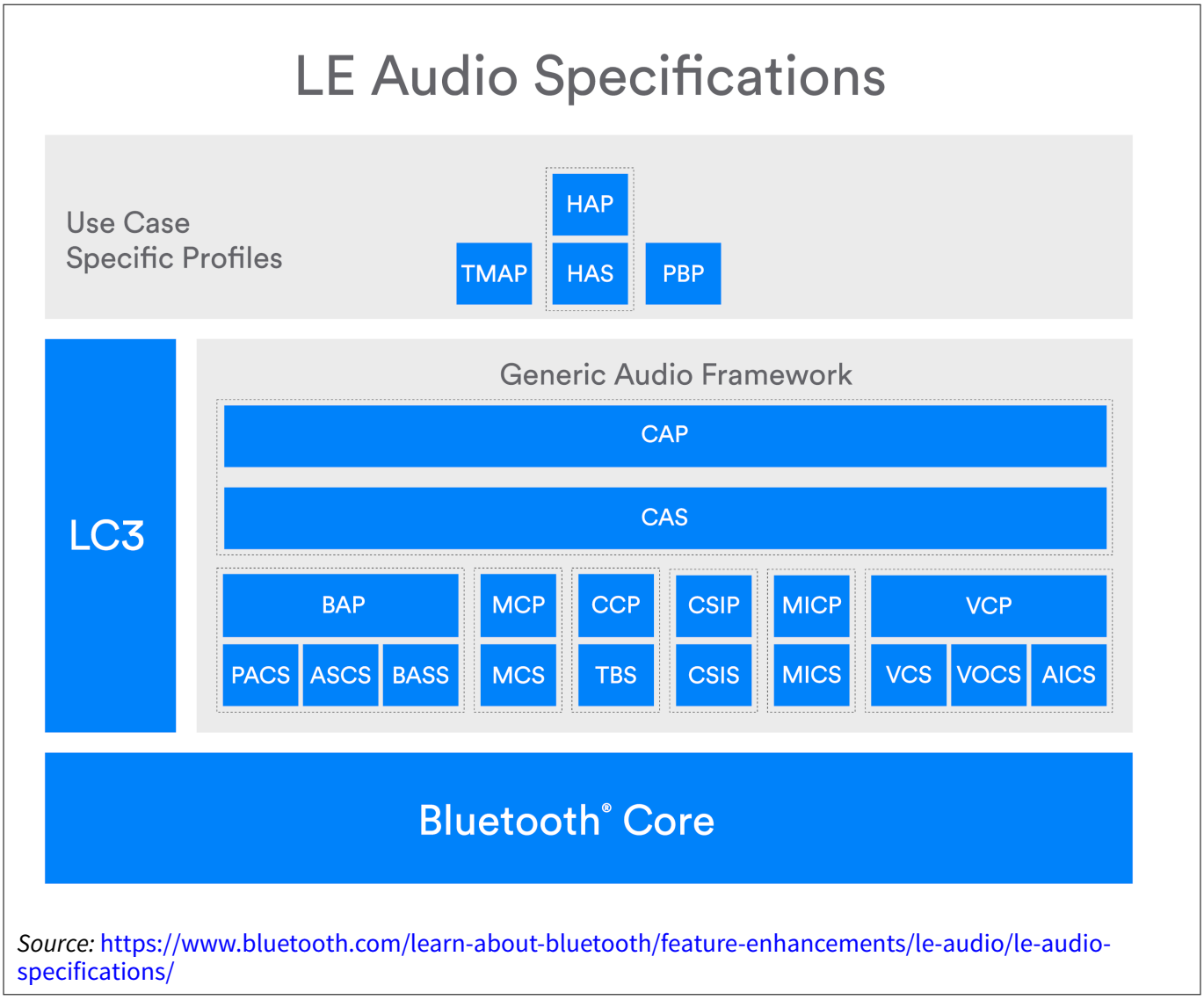
2.2. Bluetooth® Low Energy Audio Profiles

Introduction to Bluetooth® Low Energy Audio

Bluetooth® Low Energy Audio (BLE Audio) is a significant advancement in the Bluetooth® technology ecosystem, tailored for highly efficient and low-power audio streaming. It is designed to provide robust, high-quality audio experiences while maintaining the power efficiency that BLE is known for. This technology is particularly beneficial for various audio and voice applications, including hearing aids, wireless earbuds, and broadcasting audio in public places. BLE Audio introduces several new features, such as multi-stream audio, improved audio quality, and enhanced power efficiency, making it a versatile solution for modern audio applications.

Profiles Used in BLE Audio

The high-level specifications of BLE Audio along with the profiles are shown in Figure 22.



Source: https://www.bluetooth.com/learn-about-bluetooth/feature-enhancements/le-audio/le-audio-specifications/

Figure 22 Bluetooth® LE Audio Specifications

The Generic Audio Framework (GAF) is considered the middleware of the Bluetooth® LE Audio architecture. The GAF contains the profiles and services that allows higher layer applications and profiles to set up streams, change volume, control media and telephony and more. The GAF builds on GATT, GAP and isochronous channels (ISO).

GAF uses GAP to connect, advertise and synchronize to other devices. GAF uses GATT to configure streams, associate streams with content (e.g. media or telephony), control volume and more. GAF uses ISO for the audio streams themselves, both as unicast (connected) audio streams or broadcast (unconnected) audio streams.

GAF mandates the use of the LC3 codec, but also supports other codecs.

The top-level profiles TMAP and HAP are not part of the GAF, but rather provide top-level requirements for how to use the GAF.

6.2 PSOC™ Edge E84 Bluetooth® Audio & Voice Use-Cases

6.2.1 Bluetooth® Classic - A2DP Source

An A2DP Source is a device that sends a digital audio stream to an A2DP sink device over a Bluetooth®Classic connection. The audio is typically transmitted at 44.1kHz or 48kHz sampling rates in an encoded format.

An example use-case of the A2DP Source Profile on MCU is audio streaming to Bluetooth® earbuds from the MCU + AIROC™ Wi-Fi and Bluetooth® combo device - CYWxxxxx device present in a smartwatch.

The supported configurations for this MCU Bluetooth® Audio Use-Case are as follows:

- Bluetooth® Audio Streaming (A2DP)
  - Supported Sampling Rates: 44.1 kHz, 48 kHz
  - Supported Codec: SBC
- Bluetooth® Control (AVRCP)
  - Supported events:
    - Device Connected
    - Device Disconnected
    - Repeat Settings Changed
    - Shuffle Settings Changed
    - Passthrough Command - Play
    - Passthrough Command - Pause
    - Passthrough Command - Stop
    - Passthrough Command - Next Track
    - Passthrough Command - Previous Track
    - Absolute Volume Changed

Important APIs

Table 4 A2DP Source APIs

Sl. No.	API Name	API Description
1	`wiced_bt_a2dp_source_init`	API to initialize the A2DP SOURCE component and register with the stack
2	`wiced_bt_a2dp_source_connect`	API to connect to a peer device
3	`wiced_bt_a2dp_source_disconnect`	API to disconnect the connection from a connected peer device
4	`wiced_bt_a2dp_source_start`	API to start streaming. Called by the application when it wants to indicate the peer to start streaming.
5	`wiced_bt_a2dp_source_suspend`	API to suspend streaming
6	`wiced_bt_avdt_write_req`	API to send a media packet to the peer device. The stream must be started before this function is called.

High-level Code Flow

In the MCU A2DP Source use-case, the Host stack (BT Stack) and the related Bluetooth® user application is typically executed on the CM55 CPU. The detailed code flow is shown in Figure 23.

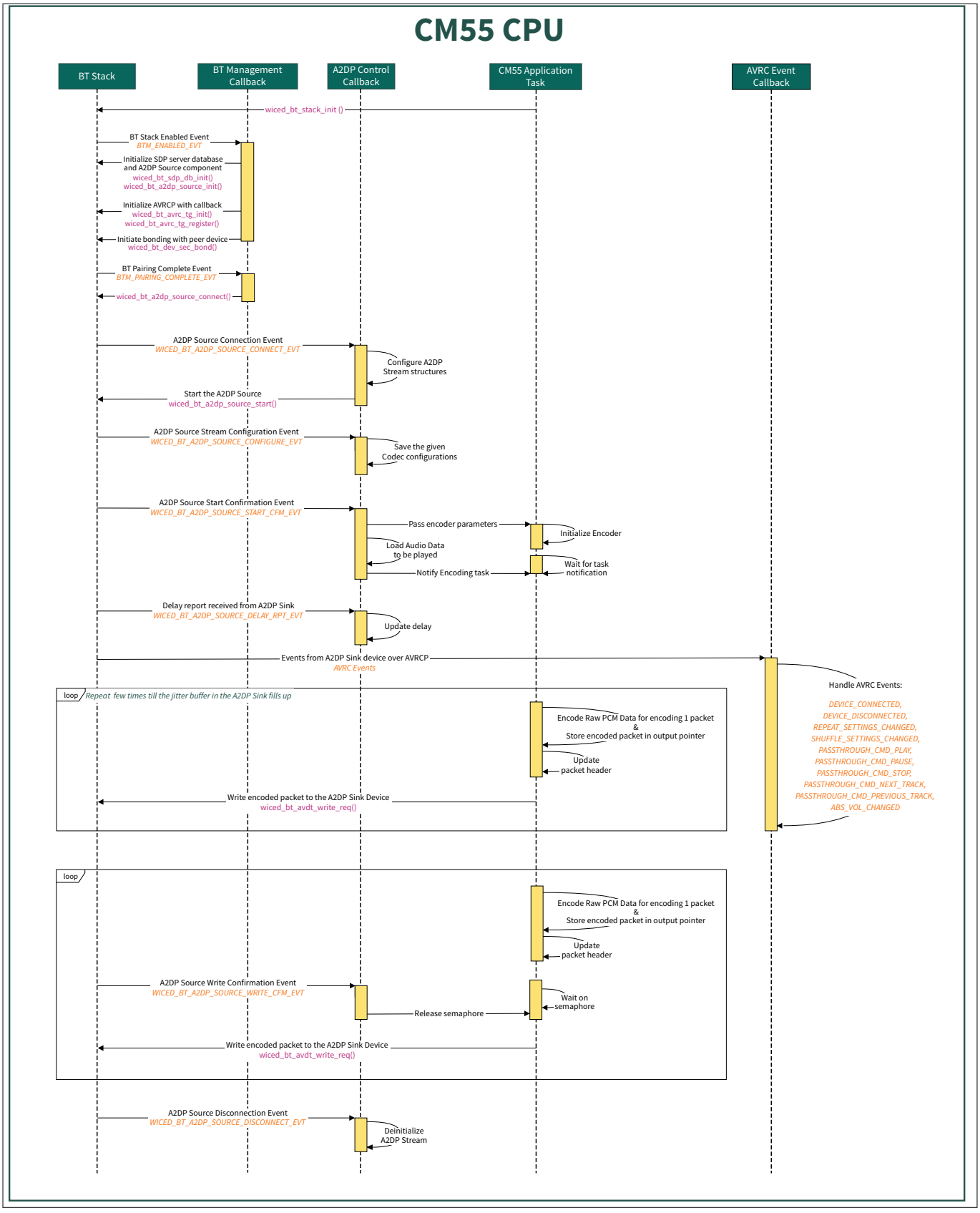


Figure 23 A2DP Source Code Flow

6.2.2 Bluetooth® Classic - A2DP Sink

The A2DP Sink is a device that acts as a sink of a digital audio stream delivered from the A2DP Source. It typically plays audio received from a source at 44.1kHz or 48kHz via a speaker attached to the A2DP Sink device.

An example use-case of the A2DP Sink Profile on MCU is audio streaming from a smartphone to the MCU + AIROC™ Wi-Fi and Bluetooth® combo device - CYWxxxxx device present in a smartwatch.

The supported configurations for this MCU Bluetooth® Audio Use-Case are as follows:

- Bluetooth® Audio Streaming (A2DP)
  - Supported Sampling Rates: 44.1 kHz, 48 kHz
  - Supported Codec: SBC
- Bluetooth® Control (AVRCP)
  - Supported events:
    - Device Connected
    - Device Disconnected
    - Repeat Settings Changed
    - Shuffle Settings Changed
    - Passthrough Command - Play
    - Passthrough Command - Pause
    - Passthrough Command - Stop
    - Passthrough Command - Next Track
    - Passthrough Command - Previous Track
    - Volume Level

Important APIs

Table 5 A2DP Sink APIs

Sl. No.	API Name	API Description
1	`wiced_bt_a2dp_sink_init`	API to initialize the A2DP SINK component and register with the stack
2	`wiced_bt_a2dp_sink_register_data_cback`	API used to register for data callback that handles the received data packets.
3	`wiced_bt_a2dp_sink_connect`	API to connect to a peer device
4	`wiced_bt_a2dp_sink_disconnect`	API to disconnect the connection from a connected peer device
5	`wiced_bt_a2dp_sink_start`	API to start streaming. Called by the application when it wants to indicate the peer to start streaming.
6	`wiced_bt_a2dp_sink_suspend`	API to suspend streaming

High-level Code Flow

In the MCU A2DP Sink use-case, the Host stack (BT Stack) and the related Bluetooth® user application is typically executed on the CM55 CPU. The detailed code flow is shown in Figure 24.

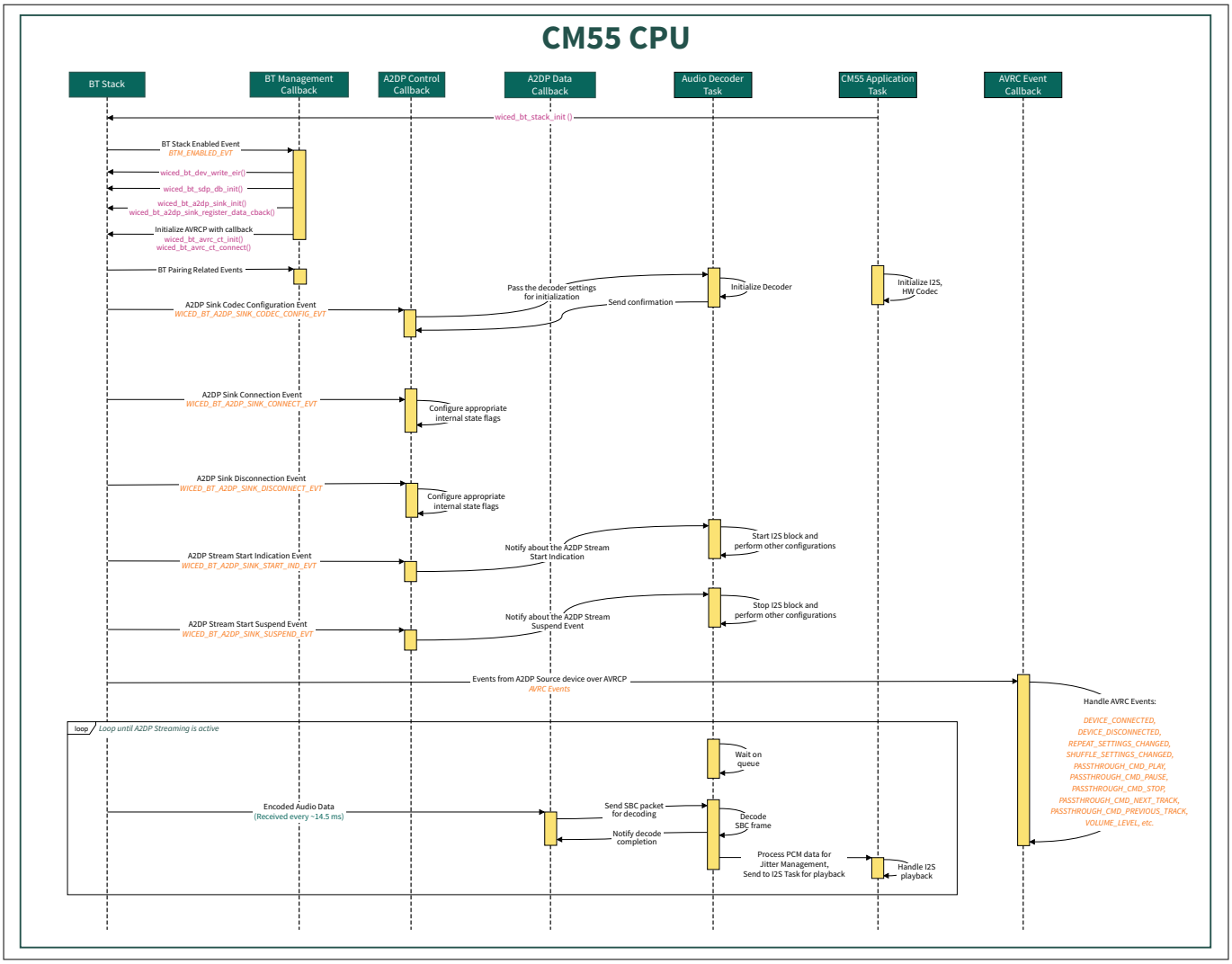


Figure 24 A2DP Sink Code Flow

6.2.3 Bluetooth® Classic - HFP

The Hands-Free Profile (HFP) is designed to provide a wireless connection between a mobile phone and a hands-free device, such as a car kit or headset. It allows for voice calls to be managed and conducted without needing to physically handle the phone, enhancing user convenience and safety, particularly while driving. In this profile, the hands-free device acts as the Hands-Free (HF) unit, while the mobile phone serves as the Audio Gateway (AG), facilitating communication between the two.

An example use-case of HFP HF Profile for the MCU applications is the voice-call capability in a smartwatch. In this use-case, the mobile phone serves as the HFP AG device and the HFP HF role is performed by the MCU + AIROC™ Wi-Fi and Bluetooth® combo device - CYWxxxxx device present in a smartwatch.

Within the scope of this document, the MCU Bluetooth® Audio capabilities include the following configurations:

- Bluetooth® Audio Call (HFP HF)
  - Supported Sampling Rates: 16 kHz
  - Supported Codec: mSBC

Important APIs

Table 6 HFP HF APIs

Sl. No.	API Name	API Description
1	`wiced_bt_hfp_hf_init`	API to initialize the HFP-HF component and register with the stack
2	`wiced_bt_sco_setup_voice_path`	API to configure the SCO data routing path
3	`wiced_bt_hfp_hf_connect`	API to initiate a HFP connection to an AG
4	`wiced_bt_hfp_hf_disconnect`	API to disconnect from an AG
5	`wiced_bt_hfp_hf_perform_call_action`	API to manipulate a call (i.e., to answer, hold, hangup, reject, etc).
6	`wiced_bt_sco_create_as_acceptor`	API to create a synchronous connection oriented connection as acceptor.
7	`wiced_bt_sco_accept_connection`	API called to handle (e)SCO connection request event
8	`wiced_bt_sco_write_buffer`	API to write SCO data to a specified instance.
9	`wiced_bt_sco_remove`	API to remove a specific synchronous connection oriented connection.

High-level Code Flow

In the MCU HFP HF use-case, the Host stack (BT Stack) and the related Bluetooth® user application is typically executed on the CM55 CPU. The detailed code flow is shown in Figure 25.

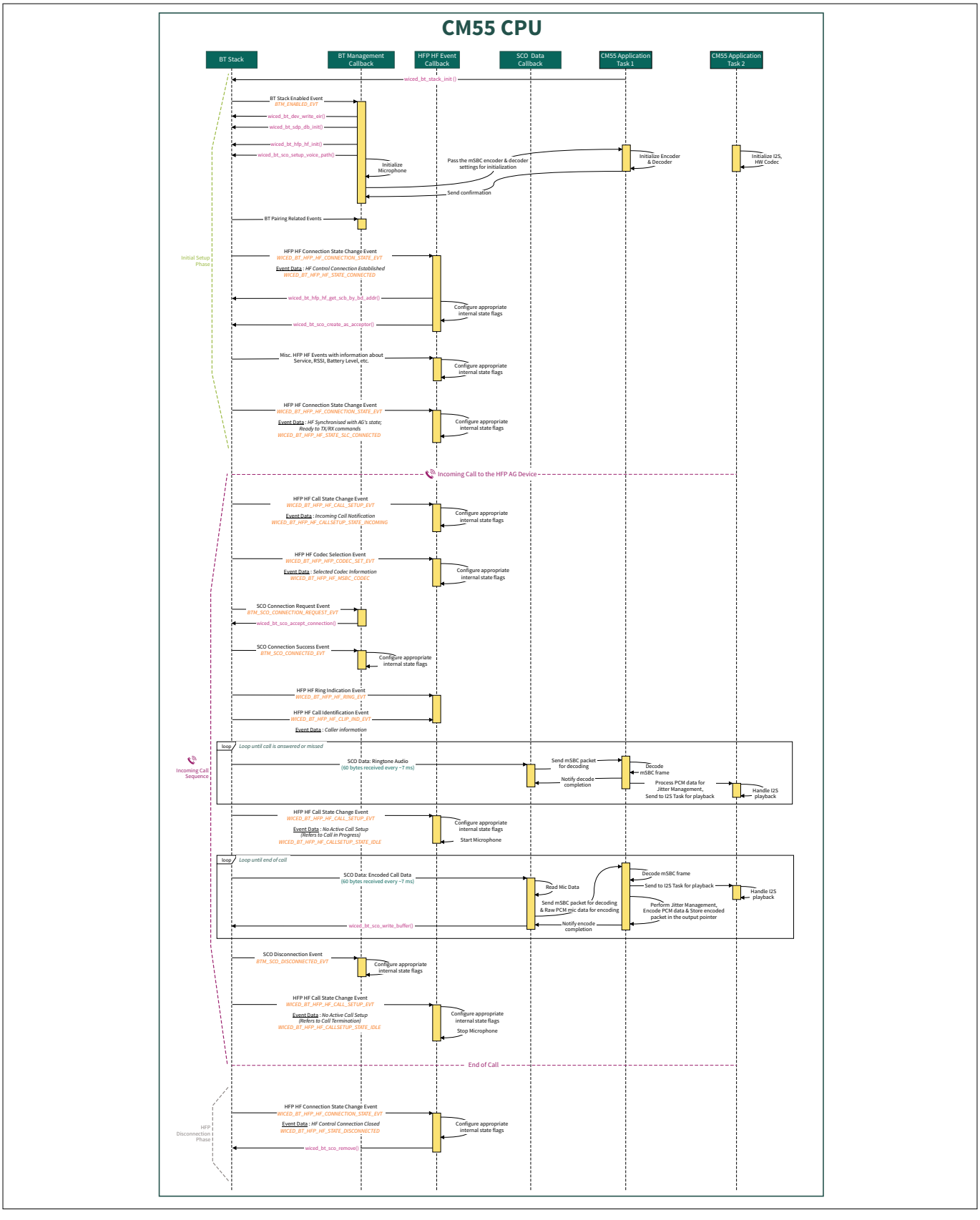


Figure 25 HFP HF Code Flow

6.2.4 Bluetooth® LE Audio - Unicast Sink

In Bluetooth® LE Audio, a Unicast Sink application refers to the scenario where a single audio source transmits audio data to a single receiving device, which acts as the sink. This is a common use case in applications such as wireless earbuds or headphones, where the compatible audio source (e.g., a smartphone or a music player) sends audio streams directly to the earbuds.

An example use-case of Unicast Sink for the MCU applications is the audio streaming from a Bluetooth® LE Audio capable Unicast Source device to the MCU + AIROC™ Wi-Fi and Bluetooth® combo device - CYWxxxxx device that acts as a Unicast Sink device.

The supported configurations for this MCU Bluetooth® Audio Use-Case are as follows:

- Bluetooth® Audio Streaming (Unicast Sink)
  - Supported Sampling Rates: 8 kHz, 16 kHz, 24 kHz, and 48 kHz
  - Supported Codec: LC3
  - Supported streaming intervals: 7.5 ms and 10 ms
- Volume Control Service (VCS)
  - Supported events:
    - Mute
    - Unmute
    - Relative Volume Down
    - Relative Volume Up
    - Unmute Relative Volume Down
    - Unmute Relative Volume Up
    - Set Absolute Volume

Important APIs

Table 7 Unicast Sink APIs

Sl. No.	API Name	API Description
1	`wiced_bt_gatt_db_init`	API to initialize the GATT database
2	`wiced_bt_gatt_register`	API to register an application callback with GATT
3	`gatt_interface_init`	API to initialize the gatt interface for BLE Audio applications
4	`wiced_bt_ga_pacs_init`	API to initialize the Published Audio Capabilities Service (PACS)
5	`wiced_bt_ga_ascs_init`	API to initialize the Audio Stream Control Service (ASCS)
6	`wiced_bt_ga_vcs_init`	API to initialize the Volume Control Service (VCS)
7	`wiced_bt_ga_mcs_init`	API to initialize the Media Control Service (MCS)
8	`wiced_bt_ga_gmcs_init`	API to initialize the Generic Media Control Service (GMCS)
9	`gatt_interface_setup_services_from_local_db`	API to setup the server services by reading the GATT DB
10	`wiced_bt_isoc_register_cb`	API to register the ISOC event callback handler

(table continues...)

Table 7 (continued) Unicast Sink APIs

Sl. No.	API Name	API Description
11	`wiced_bt_isoc_register_data_cb`	API to register the ISOC data callback handler

High-level Code Flow

In the MCU Unicast Sink use-case, the Host stack (BT Stack) and the related Bluetooth® user application is typically executed on the CM55 CPU. The detailed code flow is shown in Figure 26.

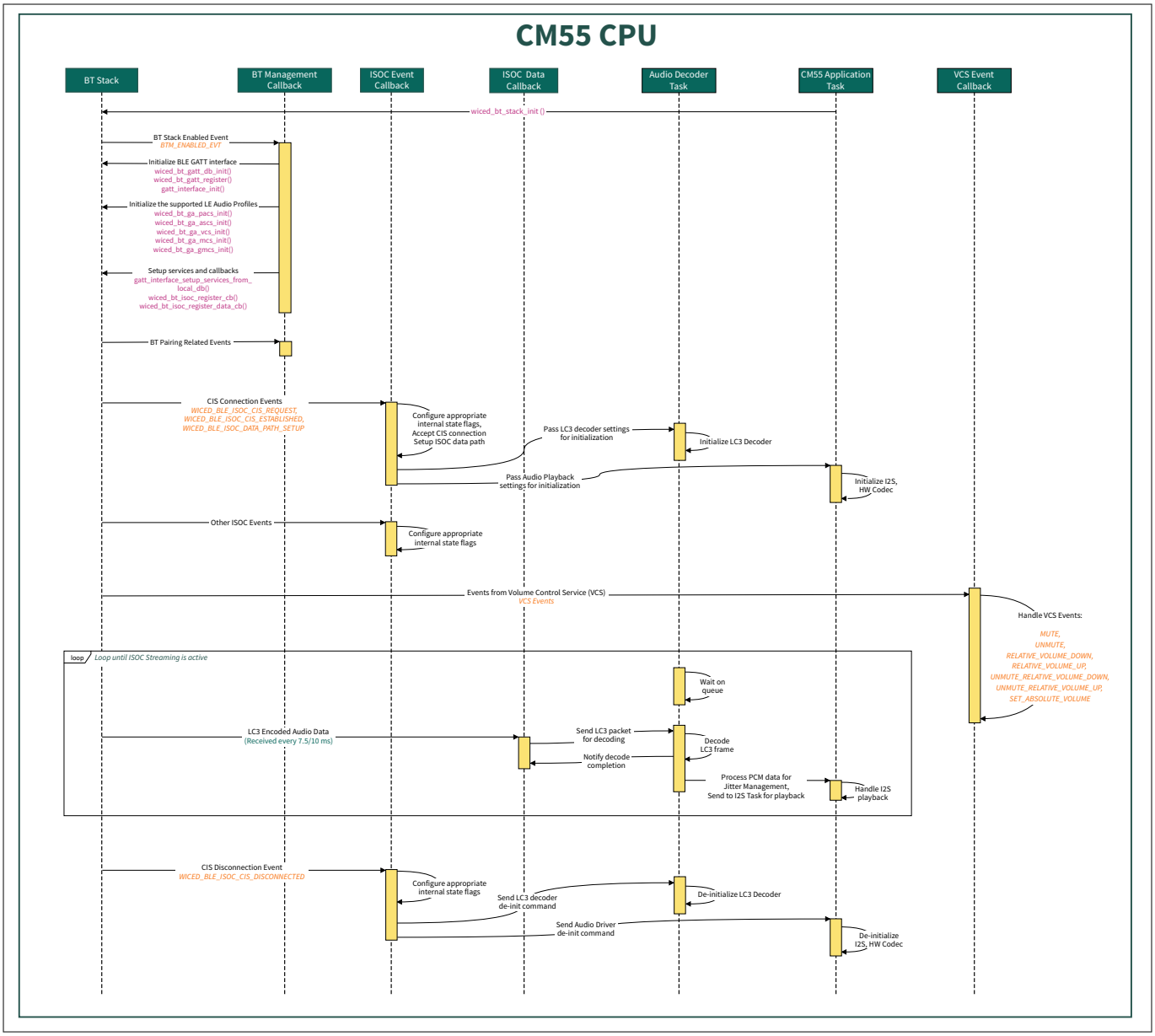


Figure 26 BLE Unicast Sink Code Flow

Revision history

Document version	Date of release	Description of changes
**	2025-10-23	Initial release

Trademarks

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-10-23

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-vft1759726486022

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.