nRF5340 Audio

The nRF5340 Audio application demonstrates audio playback over isochronous channels (ISO) using LC3 codec compression and decompression, as per Bluetooth® LE Audio specifications. It is developed for use with the nRF5340 Audio development kit.

In its default configuration, the application requires the LC3 software codec. The application also comes with various tools, including the `buildprog.py` Python script that simplifies building and programming the firmware.

Feature support matrix

The following table lists features of the nRF5340 Audio application and their respective limitations and maturity level. For an explanation of the maturity levels, see Software maturity levels.

Note

Features not listed are not supported.

Overview

The application can work as a gateway or a headset. The gateway receives the audio data from external sources (USB or I2S) and forwards it to one or more headsets. The headset is a receiver device that plays back the audio it gets from the gateway. It is also possible to enable a bidirectional mode where one gateway and one headset can send and receive audio to and from each other at the same time.

Both device types use the same code base, but different firmware, and you need both types of devices for testing the application. Gateways and headsets can both run in one of the available application modes, either the *connected isochronous stream* (CIS) mode or in the *broadcast isochronous stream* (BIS) mode. The CIS mode is the default mode of the application.

Changing configuration related to the device type and the application modes requires rebuilding the firmware and reprogramming the development kits.

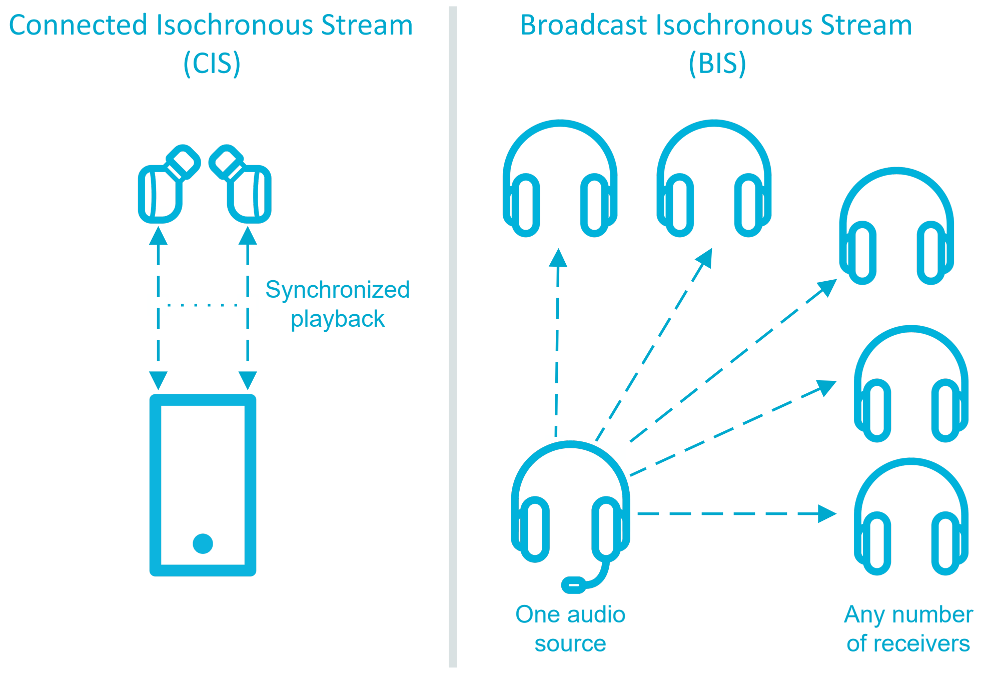
Regardless of the configuration, the application handles the audio data in the following manner:

1. The gateway receives audio data from the audio source over USB or I2S.
2. The gateway processes the audio data in its application core, which channels the data through the application layers:
   a. Audio data is sent to the synchronization module (I2S-based firmware) or directly to the software codec (USB-based firmware).
   b. Audio data is encoded by the software codec.
   c. Encoded audio data is sent to the Bluetooth LE Host.
3. The host sends the encoded audio data to the LE Audio Controller Subsystem for nRF53 on the network core.
4. The subsystem forwards the audio data to the hardware radio and sends it to the headset devices, as per the LE Audio specifications.
5. The headsets receive the encoded audio data on their hardware radio on the network core side.
6. The LE Audio Controller Subsystem for nRF53 running on each of the headsets sends the encoded audio data to the Bluetooth LE Host on the headsets' application core.
7. The headsets process the audio data in their application cores, which channel the data through the application layers:
   a. Audio data is sent to the stream control module and placed in a FIFO buffer.
   b. Audio data is sent from the FIFO buffer to the synchronization module (headsets only use I2S-based firmware).
   c. Audio data is decoded by the software codec.
8. Decoded audio data is sent to the hardware audio output over I2S.

In the I2S-based firmware for gateway and headsets, sending the audio data through the application layers includes a mandatory synchronization step using the synchronization module. This proprietary module ensures that the audio is played at the same time with the correct speed. For more information, see Synchronization module overview.

Application modes

The application can work either in the *connected isochronous stream* (CIS) mode or in the *broadcast isochronous stream* (BIS) mode, depending on the chosen firmware configuration.


Connected Isochronous Stream (CIS)
Synchronized playback
Broadcast Isochronous Stream (BIS)
One audio source
Any number of receivers

CIS and BIS mode overview

Connected Isochronous Stream (CIS)

CIS is a bidirectional communication protocol that allows for sending separate connected audio streams from a source device to one or more receivers. The gateway can send the audio data using both the left and the right ISO channels at the same time, allowing for stereophonic sound reproduction with synchronized playback.

This is the default configuration of the nRF5340 Audio application. In this configuration, you can use the nRF5340 Audio development kit in the role of the gateway, the left headset, or the right headset.

In the current version of the nRF5340 Audio application, the CIS mode offers both unidirectional and bidirectional communication. You can configure the CIS to bidirectional communication, in which it will support a walkie-talkie demonstration. In the bidirectional communication, the headset device will send audio from the on-board PDM microphone. In the walkie-talkie demonstration, the gateway device will send audio from the on-board PDM microphone instead of using the **line**-in. See Enabling the walkie-talkie demo for more information.

Note

Only one headset device can be connected when testing the bidirectional mode or the walkie-talkie demo.

Broadcast Isochronous Stream (BIS)

BIS is a unidirectional communication protocol that allows for broadcasting one or more aud streams from a source device to an unlimited number of receivers that are not connected to t source.

In this configuration, you can use the nRF5340 Audio development kit in the role of the gateway or as one of the headsets. Use multiple nRF5340 Audio development kits to test BIS having multiple receiving headsets.

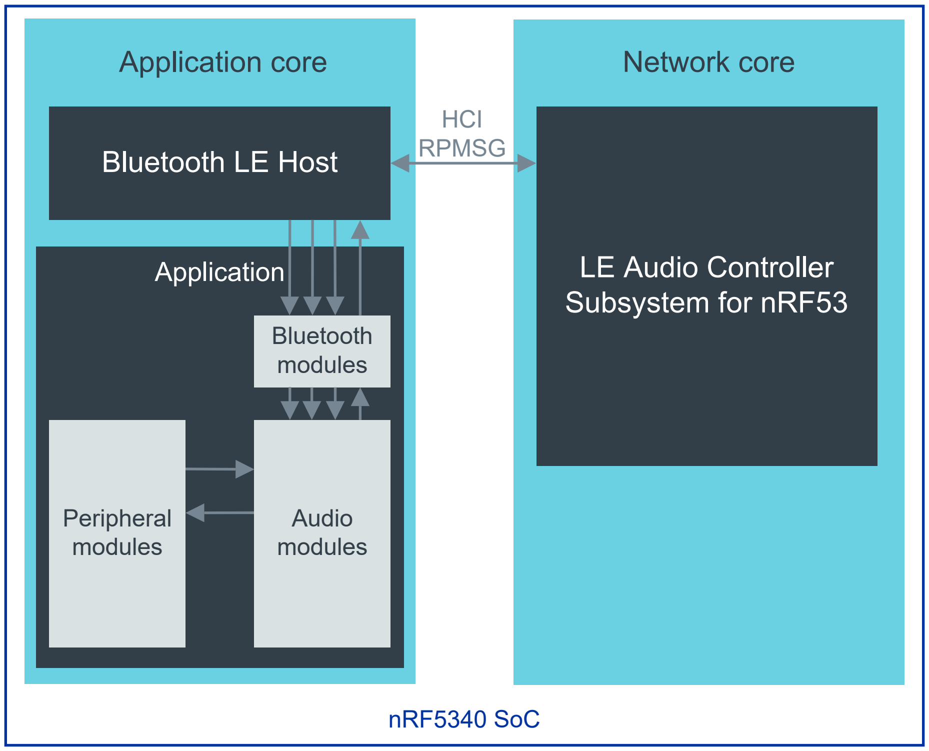
Note

In the BIS mode, you can use any number of nRF5340 Audio development kits as receivers.

The audio quality for both modes does not change, although the processing time for stereo can be longer.

Firmware architecture

The following figure illustrates the software layout for the nRF5340 Audio application:



nRF5340 Audio high-level design (overview)

The network core of the nRF5340 SoC runs the *LE Audio Controller Subsystem for nRF53*. This subsystem is a Bluetooth LE Controller that is custom-made for the application. It is responsible for receiving the audio stream data from hardware layers and forwarding the data to the Bluetooth LE host on the application core. The subsystem implements the lower layers of the Bluetooth Low Energy software stack and follows the LE Audio specification requirements.

The application core runs both the Bluetooth LE Host from Zephyr and the application layer. The application layer is composed of a series of modules from different sources. These modules include the following major ones:

- Peripheral modules from the nRF Connect SDK:

  - I2S
  - USB
  - SPI
  - TWI/I2C
  - UART (debug)
  - Timer
  - LC3 encoder/decoder
- Application-specific Bluetooth modules for handling the Bluetooth connection:
  - `le_audio_cis_gateway.c` or `le_audio_cis_headset.c` - One of these `cis` modules is used by default.
  - `le_audio_bis_gateway.c` or `le_audio_bis_headset.c` - One of these `bis` modules is selected automatically when you switch to the BIS configuration.

  Only one of these files is used at compile time. Each of these files handles the Bluetooth connection and Bluetooth events and funnels the data to the relevant audio modules.
- Application-specific custom modules:
  - Stream Control - This module implements a simple state machine for the application (`STREAMING` or `PAUSED`). It also handles events from Bluetooth LE and buttons, receives audio from the host, and forwards the audio data to the next module.
  - FIFO buffers
  - Synchronization module (part of I2S-based firmware for gateway and headsets) - See Synchronization module overview for more information.

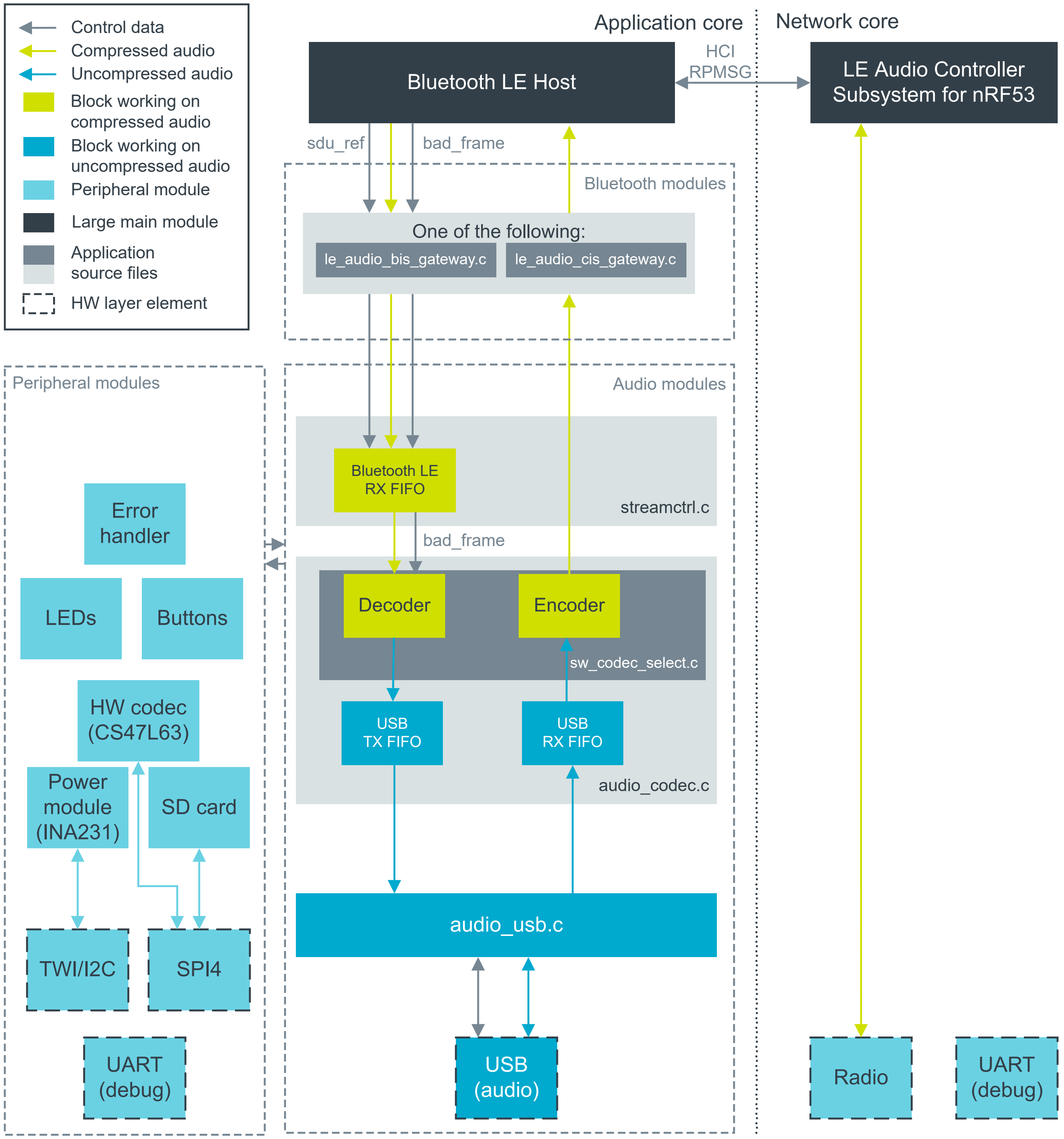
Since the application architecture is uniform and the firmware code is shared, the set of audio modules in use depends on the chosen stream mode (BIS or CIS), the chosen audio inputs and outputs (USB or analog jack), and if the gateway or the headset configuration is selected.

Note

In the current version of the application, the bootloader is disabled by default. Device Firmware Update (DFU) can only be enabled when Building and programming using script. See Configuring FOTA upgrades for details.

USB-based firmware for gateway

The following figure shows an overview of the modules currently included in the firmware that uses USB:

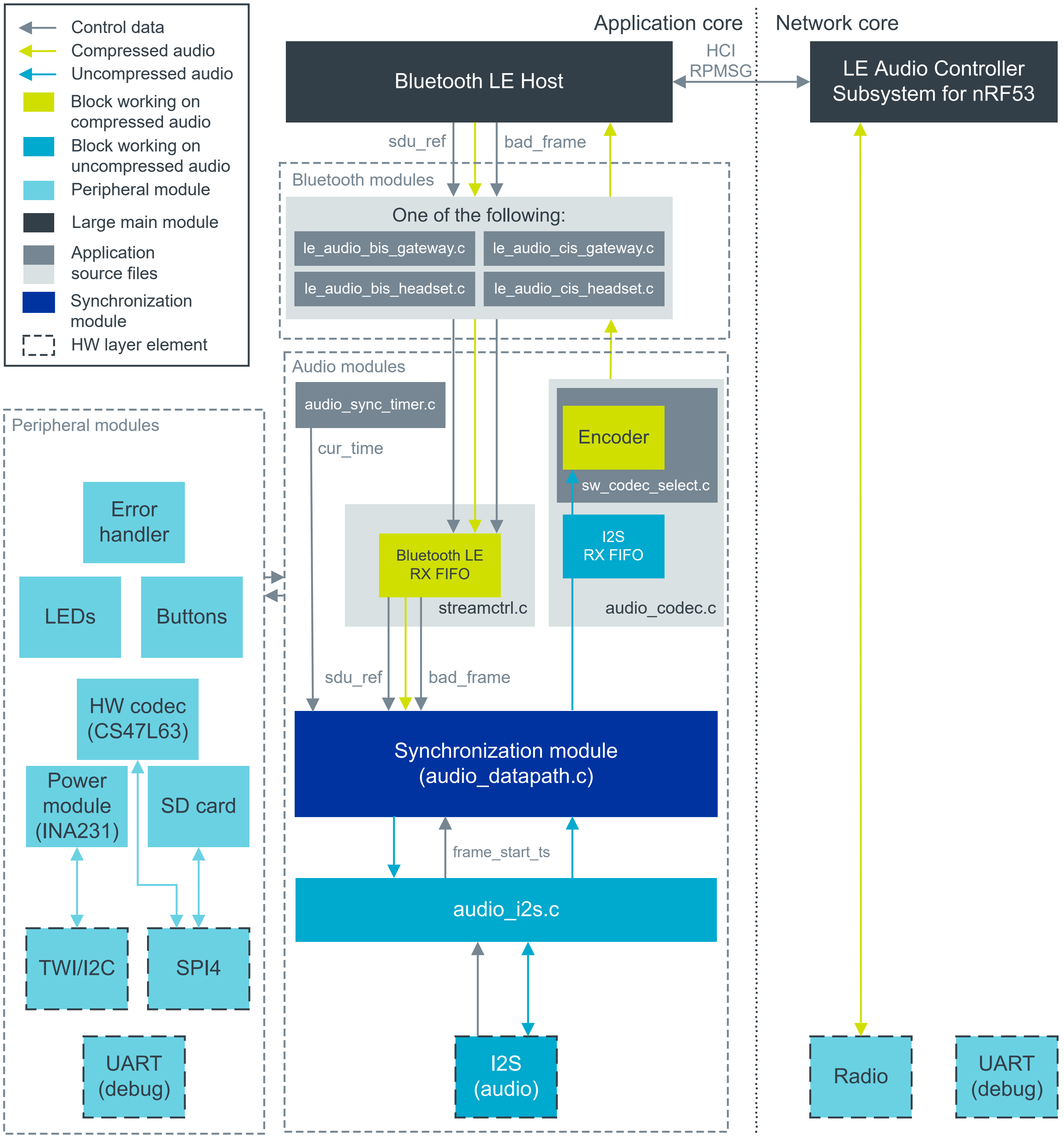


nRF5340 Audio modules on the gateway using USB

In this firmware design, no synchronization module is used after decoding the incoming frames or before encoding the outgoing ones. The Bluetooth LE RX FIFO is mainly used to make decoding run in a separate thread.

I2S-based firmware for gateway and headsets

The following figure shows an overview of the modules currently included in the firmware that u
I2S:



nRF5340 Audio modules on the gateway and the headsets using I2S

The Bluetooth LE RX FIFO is mainly used to make `audio_datapath.c` (synchronization module) run in a separate thread. After encoding the audio data received from I2S, the frames are sent by the encoder thread using a function located in `streamctrl.c`.

Synchronization module overview

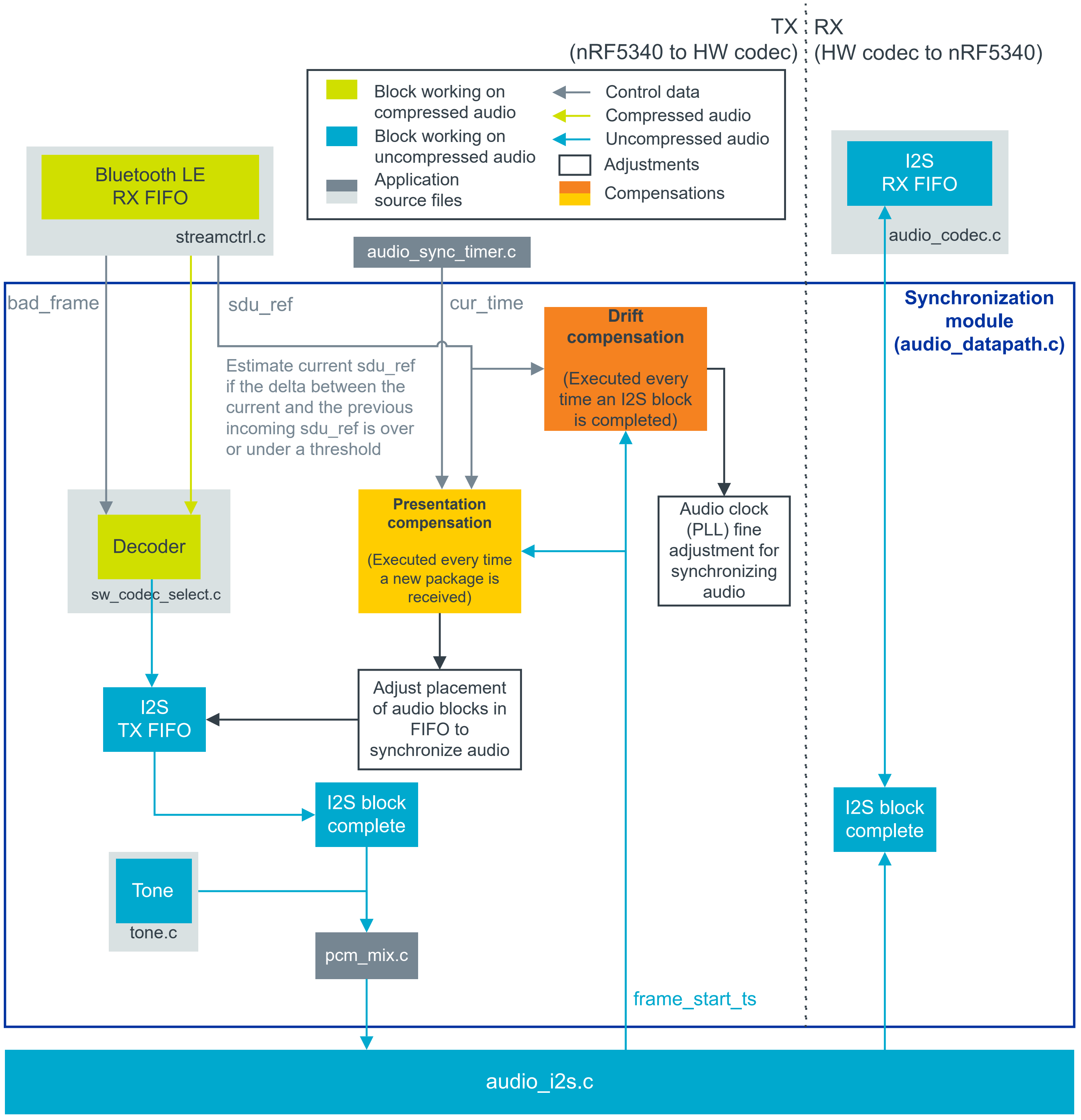
The synchronization module (`audio_datapath.c`) handles audio synchronization. To synchronize th audio, it executes the following types of adjustments:

- Presentation compensation
- Drift compensation

The presentation compensation makes all the headsets play audio at the same time, even if the packets containing the audio frames are not received at the same time on the different headsets. In practice, it moves the audio data blocks in the FIFO forward or backward a few blocks, adding blocks of *silence* when needed.

The drift compensation adjusts the frequency of the audio clock to adjust the speed at which the audio is played. This is required in the CIS mode, where the gateway and headsets must keep the audio playback synchronized to provide True Wireless Stereo (TWS) audio playback. As such, it provides both larger adjustments at the start and then continuous small adjustments to the audio synchronization. This compensation method counters any drift caused by the differences in the frequencies of the quartz crystal oscillators used in the development kits. Development kits use quartz crystal oscillators to generate a stable clock frequency. However, the frequency of these crystals always slightly differs. The drift compensation makes the inter-IC sound (I2S) interface on the headsets run as fast as the Bluetooth packets reception. This prevents I2S overruns or underruns, both in the CIS mode and the BIS mode.

See the following figure for an overview of the synchronization module.


TX
(nRF5340 to HW codec)
RX
(HW codec to nRF5340)
Block working on compressed audio
Block working on uncompressed audio
Application source files
Control data
Compressed audio
Uncompressed audio
Adjustments
Compensations
Bluetooth LE RX FIFO
streamctrl.c
audio_sync_timer.c
I2S RX FIFO
audio_codec.c
bad_frame
sdu_ref
cur_time
Synchronization module (audio_datapath.c)
Drift compensation
(Executed every time an I2S block is completed)
Estimate current sdu_ref if the delta between the current and the previous incoming sdu_ref is over or under a threshold
Decoder
sw_codec_select.c
Presentation compensation
(Executed every time a new package is received)
Audio clock (PLL) fine adjustment for synchronizing audio
I2S TX FIFO
Adjust placement of audio blocks in FIFO to synchronize audio
I2S block complete
I2S block complete
Tone
tone.c
pcm_mix.c
frame_start_ts
audio_i2s.c

nRF5340 Audio synchronization module overview

Both synchronization methods use the SDU reference timestamps (`sdu_ref`) as the reference variable. If the device is a gateway that is using I2S as audio source and the stream is unidirectional (gateway to headsets), `sdu_ref` is continuously being extracted from the LE Audio Controller Subsystem for nRF53 on the gateway. The extraction happens inside the `le_audio_cis_gateway.c` and `le_audio_bis_gateway.c` files' send function. The `sdu_ref` values are then sent to the gateway's synchronization module, and used to do drift compensation.

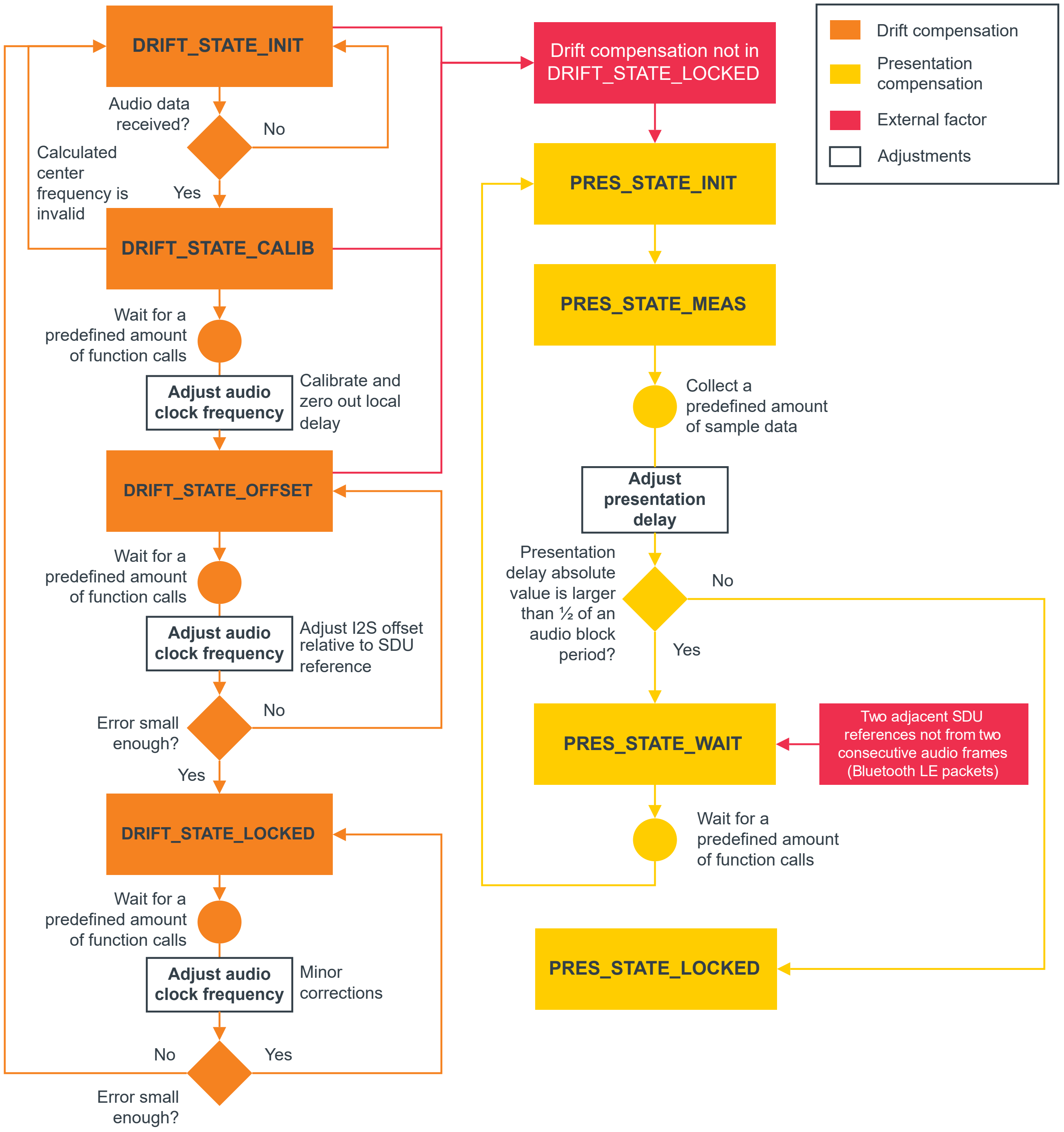
Note

Inside the synchronization module (`audio_datapath.c`), all time-related variables end with `_us` (for microseconds). This means that `sdu_ref` becomes `sdu_ref_us` inside the module.

As the nRF5340 is a dual-core SoC, and both cores need the same concept of time, each core runs a free-running timer in an infinite loop. These two timers are reset at the same time, and they run from the same clock source. This means that they should always show the same values for the same points in time. The network core of the nRF5340 running the LE controller for nRF53 uses its timer to generate the `sdu_ref` timestamp for every audio packet received. The application core running the nRF5340 Audio application uses its timer to generate `cur_time` and `frame_start_ts`.

After the decoding takes place, the audio data is divided into smaller blocks and added to a FIFO. These blocks are then continuously being fed to I2S, block by block.

See the following figure for the details of the compensation methods of the synchronization module.


Drift compensation
Presentation compensation
External factor
Adjustments
DRIFT_STATE_INIT
Audio data received?
No
Yes
Calculated center frequency is invalid
DRIFT_STATE_CALIB
Wait for a predefined amount of function calls
Adjust audio clock frequency
Calibrate and zero out local delay
DRIFT_STATE_OFFSET
Wait for a predefined amount of function calls
Adjust audio clock frequency
Adjust I2S offset relative to SDU reference
Error small enough?
No
Yes
DRIFT_STATE_LOCKED
Wait for a predefined amount of function calls
Adjust audio clock frequency
Minor corrections
No
Yes
Error small enough?
Drift compensation not in DRIFT_STATE_LOCKED
PRES_STATE_INIT
PRES_STATE_MEAS
Collect a predefined amount of sample data
Adjust presentation delay
Presentation delay absolute value is larger than ½ of an audio block period?
No
Yes
PRES_STATE_WAIT
Two adjacent SDU references not from two consecutive audio frames (Bluetooth LE packets)
Wait for a predefined amount of function calls
PRES_STATE_LOCKED

nRF5340 Audio's state machine for compensation mechanisms

The following external factors can affect the presentation compensation:

- The drift compensation must be synchronized to the locked state (`DRIFT_STATE_LOCKED`) before the presentation compensation can start. This drift compensation adjusts the frequency of the audio clock, indicating that the audio is being played at the right speed. When the drift compensation is not in the locked state, the presentation compensation does not leave the init state (`PRES_STATE_INIT`). Also, if the drift compensation loses synchronization, moving out of `DRIFT_STATE_LOCKED`, the presentation compensation moves back to `PRES_STATE_INIT`.
- When audio is being played, it is expected that a new audio frame is received in each ISO connection interval. If this does not occur, the headset might have lost its connection with the

gateway. When the connection is restored, the application receives a `sdu_ref` not consecutive with the previously received `sdu_ref`. Then the presentation compensation is put into `PRES_STATE_WAIT` to ensure that the audio is still in sync.

Note

When both the drift and presentation compensation are in state *locked* (`DRIFT_STATE_LOCKED` and `PRES_STATE_LOCKED`), **LED2** lights up.

Synchronization module flow

The received audio data in the I2S-based firmware devices follows the following path:

1. The LE Audio Controller Subsystem for nRF53 running on the network core receives the compressed audio data.
2. The controller subsystem sends the audio data to the Zephyr Bluetooth LE host similarly to the Bluetooth: HCI RPMsg sample.
3. The host sends the data to the stream control module (`streamctrl.c`).
4. The data is sent to a FIFO buffer.
5. The data is sent from the FIFO buffer to the `audio_datapath.c` synchronization module. The `audio_datapath.c` module performs the audio synchronization based on the SDU reference timestamps. Each package sent from the gateway gets a unique SDU reference timestamp. These timestamps are generated on the headset controllers (in the network core). This enables the creation of True Wireless Stereo (TWS) earbuds where the audio is synchronized in the CIS mode. It does also keep the speed of the inter-IC sound (I2S) interface synchronized with the sending and receiving speed of Bluetooth packets.
6. The `audio_datapath.c` module sends the compressed audio data to the LC3 audio decoder for decoding.
7. The audio decoder decodes the data and sends the uncompressed audio data (PCM) back to the `audio_datapath.c` module.
8. The `audio_datapath.c` module continuously feeds the uncompressed audio data to the hardware codec.
9. The hardware codec receives the uncompressed audio data over the inter-IC sound (I2S) interface and performs the digital-to-analog (DAC) conversion to an analog audio signal.

Requirements

The nRF5340 Audio application is designed to be used only with the following hardware:

Hardware platforms	PCA	Board name	Build target
nRF5340 Audio DK	PCA10121 revision 1.0.0 or above	nrf5340_audio_dk_nrf5340	`nrf5340_audio_dk_nrf5340_cpuapp`

Note

The application supports PCA10121 revisions 1.0.0 or above. The application is also compatible with the following pre-launch revisions:

- Revisions 0.8.0 and above.

You need at least two nRF5340 Audio development kits (one with the gateway firmware and one with headset firmware) to test the application. For CIS with TWS in mind, three kits are required.

Software codec requirements

The nRF5340 Audio application only supports the LC3 software codec, developed specifically for use with LE Audio.

nRF5340 Audio development kit

The nRF5340 Audio development kit is a hardware development platform that demonstrates the nRF5340 Audio application. Read the nRF5340 Audio DK Hardware documentation on Nordic Semiconductor Infocenter for more information about this development kit.

nRF5340 Audio configuration files

The nRF5340 Audio application uses `Kconfig.defaults` files to change configuration defaults automatically, based on the different application versions and device types.

Only one of the following `.conf` files is included when building:

- `prj.conf` is the default configuration file and it implements the debug application version.
- `prj_release.conf` is the optional configuration file and it implements the release application version. No debug features are enabled in the release application version. When building using the command **line**, you must explicitly specify if `prj_release.conf` is going to be included instead of `prj.conf`. See Building and running for details.

Requirements for FOTA

To test Firmware Over-The-Air (FOTA), you need an Android or iOS device with the nRF Connect

Device Manager app installed.

If you want to do FOTA upgrades for the application core and the network core at the same time, you need an external flash shield. See Configuring FOTA upgrades for more details.

User interface

The application implements a simple user interface based on the available PCB elements. You can control the application using predefined switches and buttons while the LEDs display information.

Switches

The application uses the following switches on the supported development kit:

Switch	Function
POWER	Turns the development kit on or off.
DEBUG ENABLE	Turns on or off power for debug features. This switch is used for accurate power and current measurements.

Buttons

The application uses the following buttons on the supported development kit:

Button	Function
VOL-	Depending on the moment it is pressed: • Long-pressed during startup: Changes the headset to the left channel one. • Pressed on the headset or the CIS gateway during playback: Turns the playback volume down (and unmutes).
VOL+	Depending on the moment it is pressed: • Long-pressed during startup: Changes the headset to the right channel one. • Pressed on the headset or the CIS gateway during playback: Turns the playback volume up (and unmutes).
PLAY/PAUSE	Starts or pauses the playback.

Button	Function
BTN 4	Depending on the moment it is pressed: • Long-pressed during startup: Turns on the DFU mode, if the device is configured for it. • Pressed on the gateway during playback: Sends a test tone generated on the device. Use this tone to check the synchronization of headsets. • Pressed on the gateway during playback multiple times: Changes the tone frequency. The available values are 1000 Hz, 2000 Hz, and 4000 Hz. • Pressed on a BIS headset during playback: Change audio stream, if more than one is available.
BTN 5	Depending on the moment it is pressed: • Long-pressed during startup: Clears the previously stored bonding information. • Pressed during playback: Mutes the playback volume. • Pressed on a BIS headset during playback: Change the gateway, if more than one is available.
RESET	Resets the device to the originally programmed settings. This reverts any changes made during testing, for example the channel switches with **VOL** buttons.

LEDs

To indicate the tasks performed, the application uses the LED behavior described in the following table:

LED	Indication
LED1	Off - No Bluetooth connection.
	Blinking blue - Depending on the device and the mode: • Headset: Kits have started streaming audio (BIS and CIS modes). • Gateway: Kit has connected to a headset (CIS mode) or has started broadcasting audio (BIS mode).
	Solid blue - Headset, depending on the mode: Kits have connected to the gateway (CIS mode) or found a broadcasting stream (BIS mode).
LED2	Off - Sync not achieved.
	Solid green - Sync achieved (both drift and presentation compensation are in the `LOCKED` state).
LED3	Blinking green - The nRF5340 Audio DK application core is running.
CODEC	Off - No configuration loaded to the onboard hardware codec.
	Solid green - Hardware codec configuration loaded.

LED	Indication
RGB1 (bottom side LEDs around the center opening)	Solid green - The device is programmed as the gateway.
	Solid blue - The device is programmed as the left headset.
	Solid magenta - The device is programmed as the right headset.
	Solid yellow - The device is programmed with factory firmware. It must be re-programmed as gateway or headset.
	Solid red (debug mode) - Fault in the application core has occurred. See UART log for details and use the **RESET** button to reset the device. In the release mode, the device resets automatically with no indication on LED or UART.
RGB 2	Controlled by the Bluetooth LE Controller on the network core.
	Blinking green - Ongoing CPU activity.
	Solid red - Error.
	Solid white (all colors on) - The **RGB 2** LED is not initialized by the Bluetooth LE Controller.
ERR	PMIC error or a charging error (or both).
CHG	Off - Charge completed or no battery connected.
	Solid yellow - Charging in progress.
OB/EXT	Off - No 3.3 V power available.
	Solid green - On-board hardware codec selected.
	Solid yellow - External hardware codec selected. This LED turns solid yellow also when the devices are reset, for the time then pins are floating.
FTDI SPI	Off - No data is written to the hardware codec using SPI.
	Yellow - The same SPI is used for both the hardware codec and the SD card. When this LED is yellow, the shared SPI is used by the FTDI to write data to the hardware codec.
IFMCU (bottom side)	Off - No PC connection available.
	Solid green - Connected to PC.
	Rapid green flash - USB enumeration failed.
HUB (bottom side)	Off - No PC connection available.
	Green - Standard USB hub operation.

Configuration

See Configuring your application for information about how to permanently or temporarily change the configuration.

Selecting the BIS mode

The CIS mode is the default operating mode for the application. You can switch to the BIS mode by adding the `CONFIG_TRANSPORT_BIS` Kconfig option set to `y` to the `prj.conf` file for the debug version and the `prj_release.conf` file for the release version.

Enabling the BIS mode with two gateways

In addition to the standard BIS mode with one gateway, you can also add a second gateway device that the BIS headsets can receive audio stream from. To configure the second gateway, add both the `CONFIG_TRANSPORT_BIS` and the `CONFIG_BT_AUDIO_USE_BROADCAST_NAME_ALT` Kconfig options set to `y` to the `prj.conf` file for the debug version and to the `prj_release.conf` file for the release version. You can provide an alternative name to the second gateway using the `CONFIG_BT_AUDIO_BROADCAST_NAME_ALT` or use the default alternative name.

You build each BIS gateway separately using the normal procedures from Building and running. After building the first gateway, configure the required Kconfig options for the second gateway and build the second gateway firmware. Remember to program the two firmware versions to two separate gateway devices.

Selecting the CIS bidirectional communication

The CIS unidirectional mode is the default operating mode for the application. You can switch to the bidirectional mode by adding the `CONFIG_STREAM_BIDIRECTIONAL` Kconfig option set to `y` to the `prj.conf` file (for the debug version) or to the `prj_release.conf` file (for the release version).

Enabling the walkie-talkie demo

The walkie-talkie demo is a bidirectional stream using the PDM microphone as input on each side. You can switch to using the walkie-talkie by adding the `CONFIG_WALKIE_TALKIE_DEMO` Kconfig option set to `y` to the `prj.conf` file (for the debug version) or to the `prj_release.conf` file (for the release version).

Note

Only one headset can be connected when using the bidirectional mode or the walkie-talkie demo.

Selecting the I2S serial

In the default configuration, the gateway application uses the USB serial port as the audio source. The Building and running and Testing steps also refer to using the USB serial connection.

You can switch to using the I2S serial connection by adding the `CONFIG_AUDIO_SOURCE_I2S` Kconfig option set to `y` to the `prj.conf` file for the debug version and the `prj_release.conf` file for the release version.

When testing the application, an additional audio jack cable is required to use I2S. Use this cable to connect the audio source (PC) to the analog **LINE IN** on the development kit.

Configuring FOTA upgrades

Caution

Firmware based on the nRF Connect SDK versions earlier than v2.1.0 does not support DFU. FOTA is not available for those versions.

You can test performing separate application and network core upgrades, but for production, both cores must be updated at the same time. When updates take place in the inter-core communication module (HCI RPMsg), communication between the cores will break if they are not updated together.

You can configure Firmware Over-The-Air (FOTA) upgrades to replace the applications on both the application core and the network core. The nRF5340 Audio application supports the following types of DFU flash memory layouts:

- Internal flash memory layout - which supports only single-image DFU.
- External flash memory layout - which supports multi-image DFU.

The LE Audio Controller Subsystem for nRF53 supports both the normal and minimal sizes of the bootloader. The minimal size is specified using the `CONFIG_NETBOOT_MIN_PARTITION_SIZE`.

Hardware requirements for external flash memory DFU

To enable the external flash DFU, you need an additional flash memory shield. See Requirements for external flash memory DFU in the nRF5340 Audio DK Hardware documentation in Infocenter for more information.

Enabling FOTA upgrades

The FOTA upgrades are only available when Building and programming using script. With the appropriate parameters provided, the `buildprog.py` Python script will add overlay files for the giv DFU type. To enable the desired FOTA functions:

- To define flash memory layout, include the `-m internal` parameter for the internal layout or the `-m external` parameter for the external layout.
- To use the minimal size network core bootloader, add the `-M` parameter.

For the full list of parameters and examples, see the Running the script section.

FOTA build files

The generated FOTA build files use the following naming patterns:

- For multi-image DFU, the file is called `dfu_application.zip`. This file updates two cores with one single file.
- For single-image DFU, the bin file for the application core is called `app_update.bin`. The bin file for the network core is called `net_core_app_update.bin`. In this scenario, the cores are updated one by one with two separate files in two actions.

See Output build files for more information about the image files.

Note

The network core for both gateway and headsets is programmed with the precompiled Bluetooth Low Energy Controller binary file `ble5-ctr-rpmsg_<XYZ>.hex`, where <*XYZ*> corresponds to the controller version, for example `ble5-ctr-rpmsg_3216.hex`. This file includes the LE Audio Controller Subsystem for nRF53 and is provided in the `applications/nrf5340_audio/bin` directory. If DFU is enabled, the subsystem's binary file will be generated in the `build/zephyr/` directory and will be called `net_core_app_signed.hex`.

Entering the DFU mode

The nRF Connect SDK uses SMP server and mcumgr as the DFU backend. Unlike the CIS and BIS modes for gateway and headsets, the DFU mode is advertising using the SMP server service. For this reason, to enter the DFU mode, you must long press **BTN 4** during each device startup to have the nRF5340 Audio DK enter the DFU mode.

To identify the devices before the DFU takes place, the DFU mode advertising names mention the device type directly. The names follow the pattern in which the device *ROLE* is inserted before the `_DFU` suffix. For example:

- Gateway: NRF5340_AUDIO_GW_DFU
- Left Headset: NRF5340_AUDIO_HL_DFU
- Right Headset: NRF5340_AUDIO_HR_DFU

The first part of these names is based on `CONFIG_BT_DEVICE_NAME`.

Building and running

This sample can be found under `applications/nrf5340_audio` in the nRF Connect SDK folder structure.

Note

Building and programming the nRF5340 Audio application is different from the standard procedure of building and programming for the nRF5340 DK. This is because the nRF5340 Audio application only builds and programs the files for the application core. The network core for both gateway and headsets is programmed with the precompiled Bluetooth Low Energy Controller binary file `ble5-ctr-rpmsg_<XYZ>.hex`, where *<XYZ>* corresponds to the controller version, for example `ble5-ctr-rpmsg_3216.hex`. This file includes the LE Audio Controller Subsystem for nRF53 and is provided in the `applications/nrf5340_audio/bin` directory. If DFU is enabled, the subsystem's binary file will be generated in the `build/zephyr/` directory and will be called `net_core_app_signed.hex`.

You can build and program the application in one of the following ways:

- Building and programming using script. This is the suggested method. Using this method allows you to build and program multiple development kits at the same time.
- Building and programming using command **line**. Using this method requires building and programming each development kit separately.

Note

You might want to check the nRF5340 Audio application known issues before building and programming the application.

Testing out of the box

Each development kit comes preprogrammed with basic firmware that indicates if the kit is functional. Before building the application, you can verify if the kit is working by completing the following steps:

1. Plug the device into the USB port.
2. Turn on the development kit using the On/Off switch.
3. Observe **RGB1** (bottom side LEDs around the center opening that illuminate the Nordic Semiconductor logo) turn solid yellow, **OB/EXT** turn solid green, and **LED3** start blinking

green.

You can now program the development kits with either gateway or headset firmware before they can be used.

Adding FEM support

You can add support for the nRF21540 front-end module (FEM) to this application by using one of the following options, depending on how you decide to build the application:

- If you opt for Building and programming using script, add the `--nrf21540` to the script's building command.
- If you opt for Building and programming using command **line**, add the `-DSHIELD=nrf21540_ek_fwd` to the `west build` command. For example:

```
west build -b nrf5340_audio_dk_nrf5340_cpuapp --pristine -- -DCONFIG_AUDIO_DEV=1
-DSHIELD=nrf21540_ek_fwd -DCONF_FILE=prj_release.conf
```

You can use the **`CONFIG_NRF_21540_MAIN_TX_POWER`** and **`CONFIG_NRF_21540_PRI_ADV_TX_POWER`** to set the TX power output.

See Working with RF front-end modules for more information about FEM in the nRF Connect SDK.

Building and programming using script

The suggested method for building the application and programming it to the development kit is running the `buildprog.py` Python script, which is located in the `applications/nrf5340_audio/tools/buildprog` directory. The script automates the process of selecting configuration files and building different versions of the application. This eases the process of building and programming images for multiple development kits.

Preparing the JSON file

The script depends on the settings defined in the `nrf5340_audio_dk_devices.json` file. Before using the script, make sure to update this file with the following information for each development kit you want to use:

- `nrf5340_audio_dk_snr` – This field lists the SEGGER serial number. You can check this number on the sticker on the nRF5340 Audio development kit. Alternatively, connect the development kit to a PC and run `nrfjprog -i` in a command window to print the SEGGER serial number of the kit.
- `nrf5340_audio_dk_dev` – This field assigns the specific nRF5340 Audio development kit to be a

headset or a gateway.

- `channel` – This field is valid only for headsets operating in the CIS mode. It sets the channels on which the headset is meant to work. When no channel is set, the headset is programmed as a left channel one.

Running the script

After editing the `nrf5340_audio_dk_devices.json` file, run `buildprog.py` to build the firmware for the development kits. The building command for running the script requires providing the following parameters, in **line** with nRF5340 Audio configuration files:

- Core type (`-c` parameter): `app` , `net` , or `both`
- Application version (`-b` parameter): either `release` or `debug`
- Device type (`-d` parameter): `headset` , `gateway` , or `both`
- DFU type (`-m` parameter): `internal` , `external`
- Network core bootloader minimal size (`-M`)

See the following examples of the parameter usage with the command run from the `buildprog` directory:

- Example 1: The following command builds the application using the script for the application core with the `debug` application version for both the headset and the gateway:

```
python buildprog.py -c app -b debug -d both
```

- Example 2: The following command builds the application using the script for both the application and the network core (`both`). As in *example 1*, it builds with the `debug` application version, but with the DFU internal flash memory layout enabled and using the minimal size of the network core bootloader:

```
python buildprog.py -c both -b debug -d both -m internal -M
```

  If you run this command with the `external` DFU type parameter instead of `internal` , the external flash memory layout will be enabled.

The command can be run from any location, as long as the correct path to `buildprog.py` is given.

The build files are saved in the `applications/nrf5340_audio/build` directory. The script creates a directory for each application version and device type combination. For example, when running the command above, the script creates the `dev_gateway/build_debug` and `dev_headset/build_debug` directories.

Programming with the script

The development kits are programmed according to the serial numbers set in the JSON file. Make sure to connect the development kits to your PC using USB and turn them on using the **POWER** switch before you run the command.

The following parameters are available for programming:

- Programming (`-p` parameter) – If you run the building script with this parameter, you can program one or both of the cores after building the files.
- Sequential programming (`-s` parameter) – If you are using Windows Subsystem for Linux (WSL) and encounter problems while programming, include this parameter alongside other parameters to program sequentially.

The command for programming can look as follows:

```
python buildprog.py -c both -b debug -d both -p
```

This command builds the application with the `debug` application version for both the headset and the gateway and programs the application core. Given the `-c both` parameter, it also takes the precompiled Bluetooth Low Energy Controller binary from the `applications/nrf5340_audio/bin` directory and programs it to the network core of both the gateway and the headset.

Note

If the programming command fails because of Readback protection, run `buildprog.py` with the `--recover-on-fail` or `-f` parameter to recover and re-program automatically when programming fails. For example, using the programming command example above:

```
python buildprog.py -c both -b debug -d both -p --recover-on-fail
```

If you want to program firmware that has DFU enabled, you must include the DFU parameters in the command. The command for programming with DFU enabled can look as follows:

```
python buildprog.py -c both -b debug -d both -m internal -M -p
```

Getting help

Run `python buildprog.py -h` for information about all available script parameters.

Configuration table overview

When running the script command, a table similar to the following one is displayed to provide an overview of the selected options and parameter values:

```
+------------+----------+---------+--------------+---------------------+---------------------+
| snr        | snr conn | device  | only reboot  | core app programmed | core net programmed |
+------------+----------+---------+--------------+---------------------+---------------------+
| 1010101010 | True     | headset | Not selected | Selected TBD        | Not selected        |
| 2020202020 | True     | gateway | Not selected | Selected TBD        | Not selected        |
| 3030303030 | True     | headset | Not selected | Selected TBD        | Not selected        |
+------------+----------+---------+--------------+---------------------+---------------------+
```

See the following table for the meaning of each column and the list of possible values:

Column	Indication	Possible values
`snr`	Serial number of the device, as provided in the `nrf5340_audio_dk_devices.json` file.	Serial number.
`snr conn`	Whether the device with the provided serial number is connected to the PC with a serial connection.	`True` - Connected.
		`False` - Not connected.
`device`	Device type, as provided in the `nrf5340_audio_dk_devices.json` file.	`headset` - Headset.
		`gateway` - Gateway.
`only reboot`	Whether the device is to be only reset and not programmed. This depends on the `-r` parameter in the command, which overrides other parameters.	`Not selected` - No reset.
		`Selected TBD` - Only reset requested.
		`Done` - Reset done.
		`Failed` - Reset failed.
`core app programmed`	Whether the application core is to be programmed. This depends on the value provided to the `-c` parameter (see above).	`Not selected` - Core will not be programmed.
		`Selected TBD` - Programming requested.
		`Done` - Programming done.
		`Failed` - Programming failed.

Column	Indication	Possible values
`core net programmed`	Whether the network core is to be programmed. This depends on the value provided to the `-c` parameter (see above).	`Not selected` - Core will not be programmed.
		`Selected TBD` - Programming requested.
		`Done` - Programming done.
		`Failed` - Programming failed.

Building and programming using command **line**

You can also build the nRF5340 Audio application using the standard nRF Connect SDK build steps for the command **line**.

Note

Using this method requires you to build and program each development kit one at a time before moving to the next configuration, which can be time-consuming. Building and programming using script is recommended.

Building the application

Complete the following steps to build the application:

1. Choose the combination of build flags:

    a. Choose the device type by using one of the following options:

    - For headset device: `-DCONFIG_AUDIO_DEV=1`
    - For gateway device: `-DCONFIG_AUDIO_DEV=2`

    b. Choose the application version by using one of the following options:

    - For the debug version: No build flag needed.
    - For the release version: `-DCONF_FILE=prj_release.conf`

2. Build the application using the standard build steps. For example, if you want to build the firmware for the application core as a headset using the `release` application version, you can run the following command:

```
west build -b nrf5340_audio_dk_nrf5340_cpuapp --pristine -- -DCONFIG_AUDIO_DEV=1
-DCONF_FILE=prj_release.conf
```

Unlike when Building and programming using script, this command creates the build files directly in the `build` directory. This means that you first need to program the headset development kits before you build and program gateway development kits. Alternatively, you can add the `-d` parameter to the `west` command to specify a custom build folder. This lets you build firmware for both headset and gateway before programming any development kits.

Programming the application

After building the files for the development kit you want to program, complete the following steps to program the application from the command **line**:

1. Plug the device into the USB port.

   Note

   Make sure to check the nRF5340 Audio application known issues related to serial connection with the USB.

2. Turn on the development kit using the On/Off switch.
3. Open a command prompt.
4. Run the following command to print the SEGGER serial number of your development kit:

   ```
   nrfjprog -i
   ```

   Note

   Pay attention to which device is to be programmed with the gateway HEX file and which devices are to be programmed with the headset HEX file.

5. Program the network core on the development kit by running the following command:

   ```
   nrfjprog --program bin/*.hex --chiperase --coprocessor CP_NETWORK -r
   ```

   The network core for both gateway and headsets is programmed with the precompiled Bluetooth Low Energy Controller binary file `ble5-ctr-rpmsg_<XYZ>.hex`, where *<XYZ>* corresponds to the controller version, for example `ble5-ctr-rpmsg_3216.hex`. This file includes the LE Audio Controller Subsystem for nRF53 and is provided in the `applications/nrf5340_audio/bin` directory. If DFU is enabled, the subsystem's binary file will be generated in the `build/zephyr/` directory and will be called `net_core_app_signed.hex`.

6. Program the application core on the development kit with the respective HEX file from the `build` directory by running the following command:

   ```
   nrfjprog --program build/zephyr/zephyr.hex --coprocessor CP_APPLICATION --chiperase -r
   ```

   In this command, `build/zephyr/zephyr.hex` is the HEX binary file for the application core. If a custom build folder is specified, the path to this folder must be used instead of `build/`.

7. If any device is not programmed due to Readback protection, complete the following steps:

   a. Run the following commands to recover the device:

   ```
   nrfjprog --recover --coprocessor CP_NETWORK
   nrfjprog --recover
   ```

   b. Repeat steps 5 and 6 to program both cores again.

8. When using the default CIS configuration, if you want to use two headset devices, you must also populate the UICR with the desired channel for each headset. Use the following commands, depending on which headset you want to populate:

   - Left headset:

     ```
     nrfjprog --memwr 0x00FF80F4 --val 0
     ```

   - Right headset:

     ```
     nrfjprog --memwr 0x00FF80F4 --val 1
     ```

   Select the correct board when prompted with the popup or add the `--snr` parameter followed by the SEGGER serial number of the correct board at the end of the `nrfjprog` command.

Testing

After building and programming the application, you can test it for both the CIS and the BIS modes. The following testing scenarios assume you are using USB as the audio source on the gateway. This is the default setting.

Testing the default CIS mode

Complete the following steps to test the unidirectional CIS mode for one gateway and two headset devices:

1. Make sure that the development kits are still plugged into the USB ports and are turned on.

   Note

   Make sure to check the nRF5340 Audio application known issues related to serial connection with the USB.

   After programming, **RGB2** starts blinking green on every device to indicate the ongoing CPU activity on the network core. **LED3** starts blinking green on every device to indicate the ongoing CPU activity on the application core.

2. Wait for the **LED1** on the gateway to start blinking blue. This happens shortly after programming the development kit and indicates that the gateway device is connected to at least one headset and ready to send data.
3. Search the list of audio devices listed in the sound settings of your operating system for *nRF5340 USB Audio* (gateway) and select it as the output device.
4. Connect headphones to the **HEADPHONE** audio jack on both headset devices.
5. Start audio playback on your PC from any source.
6. Wait for **LED1** to blink blue on both headsets. When they do, the audio stream has started on both headsets.

   Note

   The audio outputs only to the left channel of the audio jack, even if the given headset is configured as the right headset. This is because of the mono hardware codec chip used on the development kits. If you want to play stereo sound using one development kit, you must connect an external hardware codec chip that supports stereo.

7. Wait for **LED2** to light up solid green on the headsets to indicate that the audio synchronization is achieved.
8. Press the **VOL+** button on one of the headsets. The playback volume increases for both headsets.
9. Press the **VOL-** button on the gateway. The playback volume decreases for both headse
10. Press the **PLAY/PAUSE** button on any one of the devices. The playback stops for both headsets and the streaming state for all devices is set to paused.

11. Press the **RESET** button on the gateway. The gateway resets and the playback on the unpaused headset stops. After some time, the gateway establishes the connection with both headsets and resumes the playback on the unpaused headset.
12. Press the **PLAY/PAUSE** button on any one of the devices. The playback resumes in both headsets.
13. Press the **BTN 4** button on the gateway multiple times. For each button press, the audio stream playback is stopped and the gateway sends a test tone to both headsets. These tones can be used as audio cues to check the synchronization of the headsets.
14. Hold down the **VOL+** button and press the **RESET** button on the left headset. After startup, this headset will be configured as the right channel headset.
15. Hold down the **VOL-** button and press the **RESET** button on the left headset. After startup, this headset will go back to be configured as the left channel headset. You can also just press the **RESET** button to restore the original programmed settings.

After the kits have paired for the first time, they are now bonded. This means the Long-Term Key (LTK) is stored on each side, and that the kits will only connect to each other unless the bonding information is cleared. To clear the bonding information, press and hold **BTN 5** during boot.

When you finish testing, power off the nRF5340 Audio development kits by switching the power switch from On to Off.

Testing the BIS mode

Testing the BIS mode is identical to Testing the default CIS mode, except for the following differences:

- You must select the BIS mode manually before building the application.
- You can play the audio stream with different audio settings on the receivers. For example, you can decrease or increase the volume separately for each receiver during playback.
- When pressing the **PLAY/PAUSE** button on a headset, the streaming state only changes for that given headset.
- Pressing the **PLAY/PAUSE** button on the gateway will respectively start or stop the stream for all headsets listening in.
- Pressing the **BTN 4** button on a headset will change the active audio stream. The default configuration of the BIS mode supports two audio streams (left and right).
- Pressing the **BTN 5** button on a headset will change the gateway source for the audio stream (after Enabling the BIS mode with two gateways). If a second gateway is not present, the headset will not play audio.

Testing the walkie-talkie demo

Testing the walkie-talkie demo is identical to Testing the default CIS mode, except for the following differences:

- You must enable the Kconfig option mentioned in Enabling the walkie-talkie demo before building the application.
- Instead of controlling the playback, you can speak through the PDM microphones. The **line** is open all the time, no need to press any buttons to talk, but the volume control works as in Testing the default CIS mode.

Testing FOTA upgrades

nRF Connect Device Manager can be used for testing FOTA upgrades. The procedure for upgrading the firmware is identical for both headset and gateway firmware. You can test upgrading the firmware on both cores at the same time on a headset device by completing the following steps:

1. Make sure you have configured the application for FOTA.
2. Install nRF Connect Device Manager on your Android or iOS device.
3. Connect an external flash shield to the headset.
4. Make sure the headset runs a firmware that supports DFU using external flash memory. One way of doing this is to connect the headset to the USB port, turn it on, and then run this command:

```
python buildprog.py -c both -b debug -d headset --pristine -m external -p
```

Note

When using the FOTA related functionality in the `buildprog.py` script on Linux, the `python` command must execute Python 3.

5. Use the `buildprog.py` script to create a zip file that contains new firmware for both cores:

```
python buildprog.py -c both -b debug -d headset --pristine -m external
```

6. Transfer the generated file to your Android or iOS device, depending on the DFU scenario. See the FOTA build files section for information about FOTA file name patterns. For transfer, you can use cloud services like Google Drive for Android or iCloud for iOS.
7. Enter the DFU mode by pressing and holding down **RESET** and **BTN 4** at the same time, and then releasing **RESET** while continuing to hold down **BTN 4** for a couple more seconds.
8. Open nRF Connect Device Manager and look for `NRF5340_AUDIO_HL_DFU` in the scanned devices window. The headset is left by default.
9. Tap on **NRF5340_AUDIO_HL_DFU** and then on the downward arrow icon at the bottom of the screen.

10. In the **Firmware Upgrade** section, tap **SELECT FILE**.
11. Select the file you transferred to the device.
12. Tap **START** and check **Confirm only** in the notification.
13. Tap **START** again to start the DFU process.
14. When the DFU has finished, verify that the new application core and network core firmware works properly.

Adapting application for end products

This section describes the relevant configuration sources and lists the steps required for adapting the nRF5340 Audio application to end products.

Board configuration sources

The nRF5340 Audio application uses the following files as board configuration sources:

- Devicetree Specification (DTS) files - These reflect the hardware configuration. See Devicetree Guide for more information about the DTS data structure.
- Kconfig files - These reflect the hardware-related software configuration. See Kconfig - Tips and Best Practices for information about how to configure them.
- Memory layout configuration files - These define the memory layout of the application.

You can see the `nrf/boards/arm/nrf5340_audio_dk_nrf5340` directory as an example of how these files are structured.

For information about differences between DTS and Kconfig, see Devicetree versus Kconfig. For detailed instructions for adding Zephyr support to a custom board, see Zephyr's Board Porting Guide.

Application configuration sources

The application configuration source file defines a set of options used by the nRF5340 Audio application. This is a `.conf` file that modifies the default Kconfig values defined in the Kconfig files.

Only one `.conf` file is included at a time. The `prj.conf` file is the default configuration file and it implements the debug application version. For the release application version, you need to include the `prj_release.conf` configuration file. In the release application version no debug features should be enabled.

The nRF5340 Audio application also use several `Kconfig.defaults` files to change configuration defaults automatically, based on the different application versions and device types.

You need to edit `prj.conf` and `prj_release.conf` if you want to add new functionalities to your application, but editing these files when adding a new board is not required.

Adding a new board

Note

The first three steps of the configuration procedure are identical to the steps described in Zephyr's Board Porting Guide.

To use the nRF5340 Audio application with your custom board:

1. Define the board files for your custom board:
    a. Create a new directory in the `nrf/boards/arm/` directory with the name of the new board.
    b. Copy the nRF5340 Audio board files from the `nrf5340_audio_dk_nrf5340` directory located in the `nrf/boards/arm/` folder to the newly created directory.
2. Edit the DTS files to make sure they match the hardware configuration. Pay attention to the following elements:
    - Pins that are used.
    - Interrupt priority that might be different.
3. Edit the board's Kconfig files to make sure they match the required system configuration. For example, disable the drivers that will not be used by your device.
4. Build the application by selecting the name of the new board (for example, `new_audio_board_name`) in your build system. For example, when building from the command **line**, add `-b new_audio_board_name` to your build command.

FOTA for end products

Do not use the default MCUBoot key for end products. See Firmware updates and Signing Binaries for more information.

To create your own app that supports DFU, you can use the nRF Connect Device Manager libraries for Android and iOS.

Changing default values

Given the requirements for the Coordinated Set Identification Service (CSIS), make sure to change the Set Identity Resolving Key (SIRK) value when adapting the application.

Dependencies

This application uses the following nrfx libraries:

- `nrfx_clock.h`
- `nrfx_gpiote.h`
- `nrfx_timer.h`
- `nrfx_dppi.h`
- `nrfx_i2s.h`
- `nrfx_ipc.h`
- `nrfx_nvmc.h`

The application also depends on the following Zephyr libraries:

- Logging
- Kernel Services
- Peripherals:
    - USB device support
- Bluetooth APIs:
    - `include/bluetooth/bluetooth.h`
    - `include/bluetooth/gatt.h`
    - `include/bluetooth/hci.h`
    - `include/bluetooth/uuid.h`

Application configuration options

`CONFIG_NRF5340_AUDIO`

`(bool)` nRF5340 Audio [EXPERIMENTAL]

None

`CONFIG_AUDIO_DEV`

`(int)` Select which device type to compile for. 1=HEADSET or 2=GATEWAY

Setting this variable to 1 selects that the project is compiled as a HEADSET device. Setting to 2 will compile as a GATEWAY.

`CONFIG_TRANSPORT_BIS`

`(bool)` Use BIS (Broadcast Isochronous Stream)

None

`CONFIG_TRANSPORT_CIS`

`(bool)` Use CIS (Connected Isochronous Stream)

None

`CONFIG_REBOOT`

`(bool)`

None

`CONFIG_MAIN_THREAD_PRIORITY`

`(int)`

None

`CONFIG_MAIN_STACK_SIZE`

`(int)`

None

`CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE`

`(int)`

None

`CONFIG_THREAD_NAME`

`(bool)`

None

`CONFIG_NCS_INCLUDE_RPMSG_CHILD_IMAGE`

`(unknown)`

None

`CONFIG_RESET_ON_FATAL_ERROR`

`(bool)`

None

`CONFIG_HW_CODEC_CIRRUS_LOGIC`

`(bool)`

None

`CONFIG_BT`

`(bool)`

None

`CONFIG_BOARD_ENABLE_DCDC_APP`

`(bool)`

None

`CONFIG_BOARD_ENABLE_DCDC_NET`

`(bool)`

None

`CONFIG_BOARD_ENABLE_CPUNET`

`(bool)`

None

`CONFIG_NFCT_PINS_AS_GPIOS`

`(bool)`

None

`CONFIG_SENSOR`

`(bool)`

None

`CONFIG_CONTIN_ARRAY`

`(bool)`

None

`CONFIG_NRFX_I2S`

`(bool)`

None

`CONFIG_PCM_MIX`

`(bool)`

None

`CONFIG_TONE`

`(bool)`

None

`CONFIG_PSCM`

`(bool)`

None

`CONFIG_DATA_FIFO`

`(bool)`

None

`CONFIG_NRFX_CLOCK`

`(bool)`

None

`CONFIG_I2C`

`(bool)`

None

`CONFIG_FPU`

`(bool)`

None

`CONFIG_FPU_SHARING`

`(bool)`

None

`CONFIG_DISK_DRIVERS`

`(bool)`

None

`CONFIG_DISK_DRIVER_SDMMC`

`(bool)`

None

`CONFIG_SPI`

`(bool)`

None

CONFIG_ADC

`(bool)`

None

CONFIG_SPI_NRFX_RAM_BUFFER_SIZE

`(int)`

None

CONFIG_FILE_SYSTEM

`(bool)`

None

CONFIG_FAT_FILESYSTEM_ELM

`(bool)`

None

CONFIG_FS_FATFS_LFN

`(bool)`

None

CONFIG_FS_FATFS_EXFAT

`(bool)`

None

CONFIG_FS_FATFS_MAX_LFN

`(int)`

None

CONFIG_NRFX_TIMER1

`(bool)`

None

`CONFIG_NRFX_DPPI`

`(bool)`

None

`CONFIG_CMSIS_DSP`

`(bool)`

None

`CONFIG_CMSIS_DSP_FASTMATH`

`(bool)`

None

`CONFIG_LC3_ENC_CHAN_MAX`

`(int)`

None

`CONFIG_LC3_DEC_CHAN_MAX`

`(int)`

None

`CONFIG_AUDIO_TEST_TONE`

`(bool)`

None

`CONFIG_AUDIO_MUTE`

`(bool)`

None

`CONFIG_AUDIO_FRAME_DURATION_7_5_MS`

`(bool)` 7.5 ms

None

`CONFIG_AUDIO_FRAME_DURATION_10_MS`

`(bool)` 10 ms

None

`CONFIG_AUDIO_FRAME_DURATION_US`

`(int)`

Audio frame duration in µs.

`CONFIG_AUDIO_SAMPLE_RATE_16000_HZ`

`(bool)` 16 kHz

Sample rate of 16kHz is currently only valid for I2S/**line**-in.

`CONFIG_AUDIO_SAMPLE_RATE_24000_HZ`

`(bool)` 24 kHz

Sample rate of 24kHz is currently only valid for I2S/**line**-in.

`CONFIG_AUDIO_SAMPLE_RATE_48000_HZ`

`(bool)` 48 kHz

Sample rate of 48kHz is valid for both I2S/**line**-in and USB.

`CONFIG_AUDIO_SAMPLE_RATE_HZ`

`(int)`

I2S supports 16, 24, and 48 kHz sample rates for both input and output. USB supports only 48 kHz for input.

`CONFIG_AUDIO_BIT_DEPTH_16`

`(bool)` 16 bit audio

None

`CONFIG_AUDIO_BIT_DEPTH_32`

`(bool)` 32 bit audio

None

`CONFIG_AUDIO_BIT_DEPTH_BITS`

`(int)`

Bit depth of one sample in storage.

`CONFIG_AUDIO_BIT_DEPTH_OCTETS`

`(int)`

Bit depth of one sample in storage given in octets.

`CONFIG_AUDIO_SOURCE_USB`

`(bool)` Use USB as audio source

Set USB as audio source. Note that this forces the stream to be unidirectional because of CPU load.

`CONFIG_AUDIO_SOURCE_I2S`

`(bool)` Use I2S as audio source

None

`CONFIG_AUDIO_HEADSET_CHANNEL_RUNTIME`

`(bool)` Select at runtime

Make channel selection at runtime. Selected value is stored in persistent memory. Left channel: Hold volume-down button on headset while resetting headset. Right channel: Hold volume-up button on headset while resetting headset.

`CONFIG_AUDIO_HEADSET_CHANNEL_COMPILE_TIME`

`(bool)` Set at compile-time

Set channel selection at compile-time.

`CONFIG_AUDIO_HEADSET_CHANNEL`

`(int)` Audio channel used by headset

Audio channel compile-time selection. Left = 0. Right = 1.

`CONFIG_SW_CODEC_LC3`

`(bool)` LC3

LC3 is the mandatory codec for LE Audio.

`CONFIG_SW_CODEC_PLC_DISABLED`

`(bool)` Skip PLC on a bad frame and fill the output buffer(s) with zeros instead

None

`CONFIG_LC3_BITRATE_MAX`

`(int)` Max bitrate for LC3

None

`CONFIG_LC3_BITRATE_MIN`

`(int)` Min bitrate for LC3

None

`CONFIG_LC3_BITRATE`

`(int)`

None

`CONFIG_BUF_BLE_RX_PACKET_NUM`

`(int)`

Value can be adjusted to affect the overall latency. This adjusts the number packets in the BLE FIFO RX buffer, which is where the main latency resides. A low value will decrease latency and reduce stability, and vice-versa. Two is recommended minimum to reduce the likelyhood of audio gaps due to BLE retransmits.

`CONFIG_STREAM_BIDIRECTIONAL`

`(bool)` Bidirectional stream

Bidirectional stream enables encoder and decoder on both sides, and one device can both send and receive audio.

`CONFIG_WALKIE_TALKIE_DEMO`

`(bool)` Walkie talkie demo

The walkie talkie demo will set up a bidirectional stream using PDM microphones on each side.

`CONFIG_MONO_TO_ALL_RECEIVERS`

`(bool)` Send mono (first/left channel) to all receivers

With this flag set, the gateway will encode and send the same (first/left) channel on all ISO channels.

`CONFIG_ENCODER_THREAD_PRIO`

`(int)` Priority for encoder thread

This is a preemptible thread.

`CONFIG_AUDIO_DATAPATH_THREAD_PRIO`

`(int)` Priority for audio datapath thread

This is a preemptible thread.

`CONFIG_ENCODER_STACK_SIZE`

`(int)` Stack size for encoder thread

None

`CONFIG_AUDIO_DATAPATH_STACK_SIZE`

`(int)` Stack size for audio datapath thread

None

`CONFIG_BT_AUDIO`

`(unknown)`

None

`CONFIG_BT_ATT_ENFORCE_FLOW`

`(unknown)`

None

`CONFIG_BT_HCI_VS_EXT`

`(unknown)`

None

`CONFIG_BT_EXT_ADV`

`(unknown)`

None

`CONFIG_BT_DEVICE_NAME`

`(unknown)`

None

`CONFIG_BT_GATT_CLIENT`

`(unknown)`

None

`CONFIG_BT_BONDABLE`

`(unknown)`

None

`CONFIG_BT_PRIVACY`

`(unknown)`

None

`CONFIG_BT_SCAN_WITH_IDENTITY`

`(unknown)`

None

`CONFIG_BT_SMP`

`(unknown)`

None

`CONFIG_BT_GAP_AUTO_UPDATE_CONN_PARAMS`

`(unknown)`

None

`CONFIG_BT_AUTO_PHY_UPDATE`

`(unknown)`

None

`CONFIG_BT_AUTO_DATA_LEN_UPDATE`

`(unknown)`

None

`CONFIG_BT_L2CAP_TX_BUF_COUNT`

`(unknown)`

None

`CONFIG_BT_BUF_ACL_RX_SIZE`

`(int)`

None

`CONFIG_SETTINGS`

`(unknown)`

None

`CONFIG_BT_SETTINGS`

`(unknown)`

None

`CONFIG_FLASH`

`(unknown)`

None

`CONFIG_FLASH_MAP`

`(unknown)`

None

`CONFIG_NVS`

`(unknown)`

None

`CONFIG_NVS_LOG_LEVEL`

`(unknown)`

None

`CONFIG_BT_AUDIO_UNICAST_SERVER`

`(unknown)`

None

`CONFIG_BT_MAX_CONN`

`(unknown)`

None

`CONFIG_BT_ISO_MAX_CHAN`

`(unknown)`

None

`CONFIG_BT_ASCS_ASE_SNK_COUNT`

`(unknown)`

None

`CONFIG_BT_ASCS_ASE_SRC_COUNT`

`(unknown)`

None

`CONFIG_BT_PERIPHERAL`

`(unknown)`

None

`CONFIG_BT_DEVICE_APPEARANCE`

`(unknown)`

None

CONFIG_BT_GAP_PERIPHERAL_PREF_PARAMS

`(unknown)`

None

CONFIG_BT_VCP_VOL_REND

`(unknown)`

None

CONFIG_BT_MCC

`(unknown)`

None

CONFIG_BT_PACS_SNK_CONTEXT

`(unknown)`

None

CONFIG_BT_PACS_SRC_CONTEXT

`(unknown)`

None

CONFIG_BT_CSIP_SET_MEMBER

`(bool)`

None

CONFIG_BT_CAP_ACCEPTOR

`(bool)`

None

`CONFIG_BT_CAP_ACCEPTOR_SET_MEMBER`

`(bool)`

None

`CONFIG_BT_AUDIO_UNICAST_CLIENT`

`(unknown)`

None

`CONFIG_BT_ISO_TX_BUF_COUNT`

`(unknown)`

None

`CONFIG_BT_MAX_PAIRED`

`(unknown)`

None

`CONFIG_BT_AUDIO_UNICAST_CLIENT_GROUP_STREAM_COUNT`

`(unknown)`

None

`CONFIG_BT_AUDIO_UNICAST_CLIENT_ASE_SNK_COUNT`

`(unknown)`

None

`CONFIG_BT_AUDIO_UNICAST_CLIENT_ASE_SRC_COUNT`

`(unknown)`

None

`CONFIG_BT_VCP_VOL_CTLR`

`(unknown)`

None

`CONFIG_BT_MCS`

`(unknown)`

None

`CONFIG_BT_GATT_DYNAMIC_DB`

`(unknown)`

None

`CONFIG_UTF8`

`(unknown)`

None

`CONFIG_BT_MPL`

`(unknown)`

None

`CONFIG_MCTL`

`(unknown)`

None

`CONFIG_MCTL_LOCAL_PLAYER_CONTROL`

`(unknown)`

None

`CONFIG_MCTL_LOCAL_PLAYER_REMOTE_CONTROL`

`(unknown)`

None

`CONFIG_BT_OBSERVER`

`(unknown)`

None

`CONFIG_BT_ISO_SYNC_RECEIVER`

`(unknown)`

None

`CONFIG_BT_AUDIO_BROADCAST_SINK`

`(unknown)`

None

`CONFIG_BT_AUDIO_BROADCAST_SNK_STREAM_COUNT`

`(unknown)`

None

`CONFIG_BT_PAC_SNK`

`(unknown)`

None

`CONFIG_BT_AUDIO_BROADCAST_NAME`

`(string)`

None

`CONFIG_BT_ISO_BROADCASTER`

`(unknown)`

None

`CONFIG_BT_AUDIO_BROADCAST_SOURCE`

`(unknown)`

None

`CONFIG_BT_AUDIO_BROADCAST_SRC_STREAM_COUNT`

`(unknown)`

None

`CONFIG_BLE_ACL_CONN_INTERVAL`

`(int)` Bluetooth LE ACL Connection Interval (x*1.25ms)

None

`CONFIG_BLE_ACL_SLAVE_LATENCY`

`(int)` Bluetooth LE Slave Latency

None

`CONFIG_BLE_ACL_SUP_TIMEOUT`

`(int)` Bluetooth LE Supervision Timeout (x*10ms)

None

`CONFIG_BLE_LE_POWER_CONTROL_ENABLED`

`(bool)` Enable LE power control feature

The LE power control feature makes devices be able to change TX power dynamically and automatically during connection, which provides effective communication.

`CONFIG_BLE_CONN_TX_POWER_0DBM`

`(bool)` 0dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_1DBM`

`(bool)` -1dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_2DBM`

`(bool)` -2dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_3DBM`

`(bool)` -3dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_4DBM`

`(bool)` -4dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_5DBM`

`(bool)` -5dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_6DBM`

`(bool)` -6dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_7DBM`

`(bool)` -7dBm

None

`CONFIG_BLE_CONN_TX_POWER_NEG_8DBM`

`(bool)` -8dBm

None

CONFIG_BLE_CONN_TX_POWER_NEG_12DBM

`(bool)` -12dBm

None

CONFIG_BLE_CONN_TX_POWER_NEG_16DBM

`(bool)` -14dBm

None

CONFIG_BLE_CONN_TX_POWER_NEG_20DBM

`(bool)` -20dBm

None

CONFIG_BLE_CONN_TX_POWER_NEG_40DBM

`(bool)` -40dBm

None

CONFIG_BLE_CONN_TX_POWER_DBM

`(int)`

None

CONFIG_BLE_ADV_TX_POWER_0DBM

`(bool)` 0dBm

None

CONFIG_BLE_ADV_TX_POWER_NEG_1DBM

`(bool)` -1dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_2DBM`

`(bool)` -2dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_3DBM`

`(bool)` -3dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_4DBM`

`(bool)` -4dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_5DBM`

`(bool)` -5dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_6DBM`

`(bool)` -6dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_7DBM`

`(bool)` -7dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_8DBM`

`(bool)` -8dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_12DBM`

`(bool)` -12dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_16DBM`

`(bool)` -14dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_20DBM`

`(bool)` -20dBm

None

`CONFIG_BLE_ADV_TX_POWER_NEG_40DBM`

`(bool)` -40dBm

None

`CONFIG_BLE_ADV_TX_POWER_DBM`

`(int)`

None

`CONFIG_NRF_21540_ACTIVE`

`(bool)`

The front end module can help boost the TX power as high as 20 dBm. Usually set through buildprog, if not set there, remember to add -DSHIELD=nrf21540_ek_fwd when building app core to include the shield overlay file.

`CONFIG_NRF_21540_MAIN_DBM`

`(int)`

None

CONFIG_NRF_21540_MAIN_TX_POWER_0DBM

`(bool)` 0dBm

None

CONFIG_NRF_21540_MAIN_TX_POWER_10DBM

`(bool)` +10dBm

None

CONFIG_NRF_21540_MAIN_TX_POWER_20DBM

`(bool)` +20dBm

None

CONFIG_NRF_21540_PRI_ADV_DBM

`(int)`

None

CONFIG_NRF_21540_PRI_ADV_TX_POWER_0DBM

`(bool)` 0dBm

None

CONFIG_NRF_21540_PRI_ADV_TX_POWER_10DBM

`(bool)` +10dBm

None

CONFIG_NRF_21540_PRI_ADV_TX_POWER_20DBM

`(bool)` +20dBm

None

CONFIG_BLE_ISO_TEST_PATTERN

`(bool)` Transmit a test pattern to measure link performance

This will transmit the same amount of data as an audio stream, but the data will be an incrementing value ranging from 0-255 and repeating. Note that enabling this feature will disable the audio stream.

`CONFIG_BLE_ISO_RX_STATS_S`

`(int)` Interval in seconds to print Bluetooth LE ISO RX stats. 0 to deactivate

None

`CONFIG_BT_AUDIO_UNICAST_RECOMMENDED`

`(bool)` Recommended unicast codec capability

Recommended unicast settings for the nRF5340_audio application 48kHz, 96kbps, 2 retransmits, 20ms transport latency, and 10ms presentation delay.

`CONFIG_BT_AUDIO_UNICAST_CONFIGURABLE`

`(bool)` Configurable unicast settings

Configurable option that doesn't follow any preset. Allows for more flexibility.

`CONFIG_BT_AUDIO_UNICAST_16_2_1`

`(bool)` 16_2_1

Unicast mandatory codec capability 16_2_1 16kHz, 32kbps, 2 retransmits, 10ms transport latency, and 40ms presentation delay.

`CONFIG_BT_AUDIO_UNICAST_24_2_1`

`(bool)` 24_2_1

Unicast codec capability 24_2_1 24kHz, 48kbps, 2 retransmits, 10ms transport latency, and 40ms presentation delay.

`CONFIG_BT_AUDIO_BROADCAST_RECOMMENDED`

`(bool)` Recommended Broadcast codec capability

Recommended broadcast settings for the nRF5340_audio application 48kHz, 96kbps, 4 retransmits, 20ms transport latency, and 10ms presentation delay.

`CONFIG_BT_AUDIO_BROADCAST_CONFIGURABLE`

`(bool)` Configurable broadcast settings

Configurable option that doesn't follow any preset. Allows for more flexibility.

`CONFIG_BT_AUDIO_BROADCAST_16_2_1`

`(bool)` 16_2_1

Broadcast mandatory codec capability 16_2_1 16kHz, 32kbps, 2 retransmits, 10ms transport latency, and 40ms presentation delay.

`CONFIG_BT_AUDIO_BROADCAST_24_2_1`

`(bool)` 24_2_1

Broadcast codec capability 24_2_1 24kHz, 48kbps, 2 retransmits, 10ms transport latency, and 40ms presentation delay.

`CONFIG_BT_AUDIO_BROADCAST_16_2_2`

`(bool)` 16_2_2

Broadcast mandatory codec capability 16_2_2 16kHz, 32kbps, 4 retransmits, 60ms transport latency, and 40ms presentation delay.

`CONFIG_BT_AUDIO_BROADCAST_24_2_2`

`(bool)` 24_2_2

Broadcast codec capability 24_2_2 24kHz, 48kbps, 4 retransmits, 60ms transport latency, and 40ms presentation delay.

`CONFIG_BT_AUDIO_BROADCAST_NAME_ALT`

`(string)` Alternative broadcast name

Alternative name of the broadcast.

`CONFIG_BT_AUDIO_USE_BROADCAST_NAME_ALT`

`(bool)` Use the alternative broadcast name

Use the alternative broadcast name.

`CONFIG_BT_AUDIO_BROADCAST_ENCRYPTED`

`(bool)` Encrypted broadcast

Encrypt the broadcast to limit the connection possibilities.

`CONFIG_BT_AUDIO_BROADCAST_ENCRYPTION_KEY`

`(string)` Broadcast encryption key

Key to use for encryption and decryption; max 16 characters.

`CONFIG_BT_AUDIO_USE_BROADCAST_ID_RANDOM`

`(bool)` Use a random broadcast ID

Use a randomly generated broadcast ID.

`CONFIG_BT_AUDIO_BROADCAST_ID_FIXED`

`(hex)` Fixed broadcast ID

Fixed broadcast ID; 3 octets. Will only be used if BT_AUDIO_USE_BROADCAST_ID_RANDOM=n. Should only be used for debugging.

`CONFIG_BT_AUDIO_BROADCAST_IMMEDIATE_FLAG`

`(bool)` Immediate rendering flag

Set the immediate rendering flag.

`CONFIG_AURACAST`

`(bool)` Enable Auracast

When Auracast is enabled a Public Broadcast Announcement will be included when advertising.

`CONFIG_BT_AUDIO_PRESENTATION_DELAY_US`

`(int)` Presentation delay

Presentation delay for the ISO link.

`CONFIG_BT_AUDIO_MAX_TRANSPORT_LATENCY_MS`

`(int)` Max transport latency

Max transport latency for the ISO link.

`CONFIG_BT_AUDIO_RETRANSMITS`

`(int)` Number of retransmits

Number of retransmits for the ISO link. 2 retransmits means a total of 3 packets sent per stream.

`CONFIG_BT_AUDIO_BITRATE`

`(int)` Bitrate for ISO stream

Bitrate for the ISO stream.

`CONFIG_CS47L63_THREAD_PRIO`

`(int)` Priority for CS47L63 thread

This is a preemptible thread

`CONFIG_CS47L63_STACK_SIZE`

`(int)` Stack size for CS47L63

None

`CONFIG_USB_DEVICE_STACK`

`(bool)`

None

`CONFIG_NET_BUF`

`(bool)`

None

`CONFIG_NET_BUF_USER_DATA_SIZE`

`(int)`

None

`CONFIG_USB_DEVICE_AUDIO`

`(bool)`

None

`CONFIG_USB_DEVICE_VID`

`(hex)`

None

`CONFIG_USB_DEVICE_PID`

`(hex)`

None

`CONFIG_USB_DEVICE_PRODUCT`

`(string)`

None

`CONFIG_USB_DEVICE_MANUFACTURER`

`(string)`

None

`CONFIG_USB_DRIVER_LOG_LEVEL`

`(int)`

None

`CONFIG_USB_DEVICE_LOG_LEVEL`

`(int)`

None

`CONFIG_BUTTON_DEBOUNCE_MS`

`(int)` Button debounce time in ms

None

`CONFIG_POWER_MEAS_INTERVAL_MS`

`(int)` Power measurement interval in milliseconds

Power measurement runs continuously, this option just establishes the results polling period. Note that this value needs to be >= the configured sampling interval on the current sensor. When below, repeated measurements will be observed.

`CONFIG_POWER_MEAS_START_ON_BOOT`

`(bool)` Start power measurements for all rails on boot

This option will automatically start and periodically print the voltage, current consumption, and power usage for the following rails: VBAT, VDD1_CODEC, VDD2_CODEC, and VDD2_NRF

`CONFIG_I2S_LRCK_FREQ_HZ`

`(int)`

The sample rate of I2S. For now this is tied directly to AUDIO_SAMPLE_RATE_HZ Note that this setting is only valid in I2S master mode.

`CONFIG_I2S_CH_NUM`

`(int)`

The I2S driver itself supports both mono and stereo. Parts of the implementation are configured only stereo.

`CONFIG_POWER_MEAS_THREAD_PRIO`

`(int)` Priority for power measurement thread

This is a preemptible thread

`CONFIG_POWER_MEAS_STACK_SIZE`

`(int)` Stack size for power measurement thread

None

`CONFIG_NRFX_NVMC`

`(bool)`

None

`CONFIG_FIFO_FRAME_SPLIT_NUM`

`(int)` Number of blocks to make up one frame of audio data

Easy DMA in I2S requires two buffers to be filled before I2S transmission will begin. In order to reduce latency, an audio frame can be split into multiple blocks with this parameter. USB sends data in 1 ms blocks, so we need the split to match that. Since we set frame size to 10 ms for USB, 10 is selected as FRAME_SPLIT_NUM

`CONFIG_FIFO_TX_FRAME_COUNT`

`(int)` Max number of audio frames in TX slab

FIFO_TX is the buffer that holds decoded audio data before it is sent to either I2S or USB

`CONFIG_FIFO_RX_FRAME_COUNT`

`(int)` Max number of audio frames in RX slab

FIFO_RX is the buffer that holds uncompressed audio data coming from either I2S or USB

`CONFIG_AUDIO_DFU`

`(int)` Select which DFU type. 0=NONE, 1=Internal flash, 2=External flash

Setting this variable to 0 disables DFU. Setting this variable to 1 selects internal flash single image DFU. Setting this variable to 2 selects external flash multi images DFU.

`CONFIG_B0N_MINIMAL`

`(bool)` B0N use minimal or not

Let CMakelist choose corresponding overlay file

`CONFIG_NCS_SAMPLE_EMPTY_NET_CORE_CHILD_IMAGE`

`(bool)` Dummy Net core application

None

`CONFIG_AUDIO_DFU_ENABLE`

`(bool)`

To show warning message of EXPERIMENTAL feature DFU

`CONFIG_BOOTLOADER_MCUBOOT`

`(bool)`

None

`CONFIG_MCUMGR`

`(bool)`

None

`CONFIG_MCUMGR_CMD_OS_MGMT`

`(bool)`

None

`CONFIG_OS_MGMT_TASKSTAT`

`(bool)`

None

`CONFIG_MCUMGR_CMD_STAT_MGMT`

`(bool)`

None

`CONFIG_MCUMGR_CMD_IMG_MGMT`

`(bool)`

None

`CONFIG_MCUMGR_BUF_SIZE`

`(int)`

None

`CONFIG_MCUMGR_SMP_BT`

`(bool)`

None

`CONFIG_MCUMGR_SMP_BT_AUTHEN`

`(bool)`

None

`CONFIG_BT_L2CAP_TX_MTU`

`(int)`

None

`CONFIG_BT_BUF_ACL_TX_SIZE`

`(int)`

None

CONFIG_MCUMGR_BUF_COUNT

`(int)`

None

CONFIG_THREAD_MONITOR

`(bool)`

None

CONFIG_STATS

`(bool)`

None

CONFIG_STATS_NAMES

`(bool)`

None

CONFIG_MCUBOOT_IMAGE_VERSION

`(string)`

None

CONFIG_BT_DEVICE_NAME_DYNAMIC

`(bool)`

None

CONFIG_BT_DEVICE_NAME_GATT_WRITABLE

`(bool)`

None

CONFIG_BT_DEVICE_NAME_MAX

`(int)`

None

`CONFIG_UPDATEABLE_IMAGE_NUMBER`

`(int)`

None

`CONFIG_IMG_ERASE_PROGRESSIVELY`

`(bool)`

None

`CONFIG_PM_EXTERNAL_FLASH_MCUBOOT_SECONDARY`

`(bool)`

None

`CONFIG_SPI_NOR`

`(bool)`

None

`CONFIG_SPI_NOR_CS_WAIT_DELAY`

`(int)`

None

`CONFIG_SPI_NOR_FLASH_LAYOUT_PAGE_SIZE`

`(int)`

None

`CONFIG_MCUMGR_GRP_ZEPHYR_BASIC`

`(bool)`

None

`CONFIG_MCUMGR_GRP_BASIC_CMD_STORAGE_ERASE`

`(bool)`

None

`CONFIG_BOOT_IMAGE_ACCESS_HOOKS`

`(bool)`

None

`CONFIG_PM_OVERRIDE_EXTERNAL_DRIVER_CHECK`

`(bool)`

None

`CONFIG_PRINT_STACK_USAGE_MS`

`(int)` Print stack usage every x milliseconds

None

Disclaimers for the nRF5340 Audio application

This application and the LE Audio Controller Subsystem for nRF53 are marked as experimental. The DFU/FOTA functionality in this application is also marked as experimental.

This LE Audio link controller is tested and works in configurations used by the present reference code (for example, 2 concurrent CIS, or BIS). No other configurations than the ones used in the reference application are tested nor documented in this release.