Getting started with the X-CUBE-OPUS audio codec evaluation and profiling software expansion for STM32Cube

Introduction

The X-CUBE-OPUS expansion software package for STM32Cube runs on different STM32 MCU families and includes a firmware example that allows to easily configure and profile Opus encoder and decoder.

The expansion is built on STM32Cube software technology to ease portability across different STM32 microcontrollers.

The software comes with a sample implementation of the drivers running on NUCLEO-F413ZH, NUCLEO-H743ZI, NUCLEO-F746ZG, NUCLEO-L552ZE-Q or P-NUCLEO-WB55 development boards.

The package provides a Python application to be used together with the STM32 example to create a point-to-point bidirectional communication. The software allows the user to choose Opus settings, via an intuitive GUI, select different audio input files for profiling test and receive back results as well as encode/decode data.

The two nodes (the host with Python app and the STM32) communicate through the ST-Link Virtual COM port with a custom protocol.

The X-CUBE-OPUS expansion software package allows the user to evaluate Opus performance depending on his target application, facilitating the choice of the correct STM32 and easily integrating the audio codec.

RELATED LINKS

Visit the STM32Cube ecosystem web page on www.st.com for further information

1 X-CUBE-OPUS software expansion for STM32Cube

1.1 Overview

The X-CUBE-OPUS software package expands STM32Cube functionality.

The package key features are:

- Complete tool to evaluate and profile the advanced Opus audio codec with each possible configuration on different STM32 MCU families (ARM Cortex-M4, M7, M33)
- Firmware example that shows how to use and integrate Opus on different STM32 MCU families
- Sample application (compatible with Python 3.7) that allows to configure Opus, send audio data to STM32 Nucleo development boards and receive profiling results
- Third-party Opus v1.3.1 (downloadable from https://www.opus-codec.org): an open, royalty-free and highly versatile audio codec that is standardized by the Internet Engineering Task Force (IETF) as RFC 6716
- Custom serial protocol to allow easy communication between the STM32 Nucleo development board and the Host using dedicated commands
- Sample implementation available on NUCLEO-F413ZH, NUCLEO-H743ZI, NUCLEO-F746ZG, NUCLEO-L552ZE-Q or P-NUCLEO-WB55 development boards
- Easy portability across different MCU families, thanks to STM32Cube
- Free, user-friendly license terms

1.2 Architecture

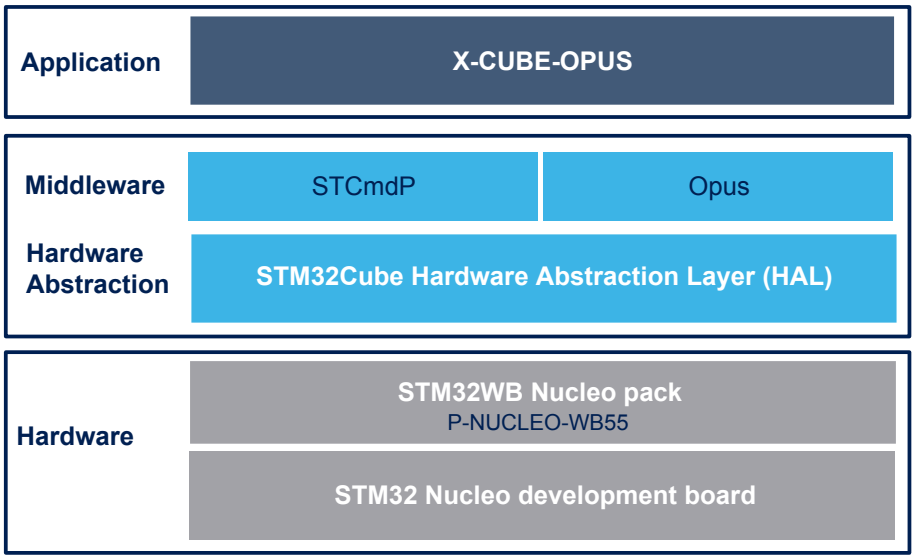
This fully compliant software expansion for STM32Cube lets you develop applications based on Opus audio codec.

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a Board Support Package (BSP) for each STM32 Nucleo development board supported, a proprietary middleware that implements a serial communication protocol and the third-party middleware Opus for audio encoding and decoding.

The software layers used by the application software to access and use the communication channel and manage user interface are:

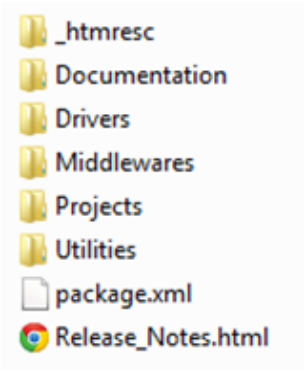
- **STM32Cube HAL layer**: the HAL driver layer provides a simple set of generic, multi-instance APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). These generic and extension APIs are directly built on a common architecture and allow overlying layers like middleware to implement their functions without depending on specific microcontroller unit (MCU) hardware information. This structure improves the library code reusability and guarantees easy portability across other devices.
- **Board support package (BSP) layer**: provides support for the peripherals on the STM32 Nucleo board (apart from the MCU). This set of APIs provides a programming interface for certain board-specific peripherals like the LED, the user button etc. This interface also helps you identify the specific board version.

Figure 1. **X-CUBE-OPUS software architecture**



1.3 Folder structure

Figure 2. **X-CUBE-OPUS package folders structure**



The following folders are included in the software package:

- **Documentation**: contains a compiled HTML file generated from the source code which details the software components and APIs.
- **Drivers**: contains the HAL drivers, the board-specific drivers for each supported board or hardware platform, including the on-board components, and the CMSIS vendor-independent hardware abstraction layer for the Cortex-M processor series.
- **Middlewares**: contains a custom command protocol for serial communication and the third-party middleware Opus for audio encoding and decoding.
- **Projects**: contains a sample application (OpusProfilingTool) provided for theNUCLEO-F413ZH, NUCLEO-H743ZI, NUCLEO-F746ZG, NUCLEO-L552ZE-Q and P-NUCLEO-WB55 platforms with three development environments (IAR Embedded Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM), and STM32CubeIDE).
- **Utilities**: contains a folder called *OPT_SW_Control* which provides a Python application. The utility lets you choose Opus settings, select different audio input file for profiling test and receive results and encode/decode data.

1.4 APIs

Detailed technical information about the APIs available to the user can be found in a compiled HTML file located inside the “Documentation” folder of the software package where all the functions and parameters are fully described.

1.5 Sample application

A sample application available for different STM32 Nucleo development boards (NUCLEO-F413ZH, NUCLEO-H743ZI, NUCLEO-F746ZG, NUCLEO-L552ZE-Q and P-NUCLEO-WB55) is provided in the “Projects” directory. Ready-to-build projects are available for multiple IDEs.

The application allows evaluating Opus performance according to the target application, facilitating the choice of the correct STM32 and easily integrating the audio codec.

A supported STM32 Nucleo board flashed with OpusProfilingTool firmware is recognized by the host device as a virtual COM port and can be connected to the *OPT_SW_Control*. The two nodes communicate using a custom serial protocol (STCmdP) to exchange configuration settings, data and results.

After the STM32Cube HAL library initialization and system clock configuration, LEDs (for user feedback) and UART (for communication) are initialized; then, in the main loop, the firmware waits for a message from the host, calling the `STCmdP_manager()` function. As soon as data are received on the UART, they are interpreted to search a complete message, using the `UART_ReceivedMSG(Msg)` function. If the message is not complete, the firmware waits for other data from the UART.

When the entire command is received, the `HandleMSG(Msg)` function is called to analyze the command and perform the relevant operation. The command response is finally prepared and sent back to the host.

1.5.1 Opus

Opus is an open, royalty-free, and highly versatile audio codec. It can be used for different kinds of applications like speech and music streaming or compressed audio storage. It is standardized by the Internet Engineering Task Force (IETF) as RFC 6716.

The scalability, from low bit-rate narrow-band speech at 6 kbit/s to very high-quality stereo music at 510 kbit/s with low complexity, makes it suitable for a wide range of interactive applications.

It consists of two layers: one based on linear prediction (LP), the other based on the modified discrete cosine transform (MDCT).

Opus efficiently combines lossless and lossy results. For example, in speech applications, LP techniques such as code-excited linear prediction (CELP) codify low frequencies more efficiently than in transform domain techniques such as MDCT.

Opus codec consists of the SILK and CELT coding technologies. The former (initially developed by Skype) uses a prediction based model (LPC) whereas the latter (from Xiph.Org) is completely modelled on the MDCT transform. This versatility allows Opus to operate in three modes (SILK, CELT or hybrid mode) and ensures multiple configurations for different applications.

Opus can handle a wide range of audio applications, including Voice over IP, videoconferencing, in-game chat, and even remote live music performances. It can scale from low bit-rate narrow-band speech to very high quality stereo music. Supported features are:

- Bit-rates from 6 kbps to 510 kbps
- Sampling rates from 8 kHz (narrow-band) to 48 kHz (full-band)
- Frame sizes from 2.5 ms to 60 ms
- Support for constant bit-rate (CBR) and variable bit-rate (VBR)
- Audio bandwidth from narrow-band to full-band
- Support for speech and music
- Support for mono and stereo
- Support for up to 255 channels (multi-stream frames)
- Dynamically adjustable bit-rate, audio bandwidth and frame size
- Good loss robustness and packet loss concealment (PLC)
- Floating point and fixed-point implementation

1.5.1.1 *Opus interface*

The X-CUBE-OPUS software expansion package provides a set of APIs to easily configure and use Opus as well as speed up the integration with your own application.

The Projects folder contains two files (opus_interface.c and opus_interface.h) to interact with the audio codec. In the header file, two configuration structures define the parameters needed to initialize an encoder and a decoder by calling `ENC_Opus_Init` and `DEC_Opus_Init` (relevant deinitialization functions are available). Depending on the chosen settings, the amount of encoded and decoded data can significantly vary; using `ENC_Opus_getMemorySize` and `DEC_Opus_getMemorySize`, it is possible to automatically calculate the memory size that has to be allocated to save the encoded and decoded buffer (considering the maximum length).

As soon as the encoder and decoder are created, it is possible to change some parameters (bit-rate value, variable or constant bit-rate, mode and complexity) and to force the use of SILK or CELT algorithm (not possible in all configurations) by calling the relevant API.

The audio frame can be encoded and decoded by calling `ENC_Opus_Encode and DEC_Opus_Decode` and giving, as parameters, the input data, the output buffer where results have to be saved and, in case of decoding, the encoded data length as it can vary frame by frame.

1.5.2 Encoder and decoder profiling

X-CUBE-OPUS software expansion package helps evaluate and measure encoder and decoder costs in terms of computational resources so that you can decide which STM32 better fits your target application according to the Opus configuration chosen.

Profiling is performed using data watchpoint trigger (DWT) STM32 registers; the clock cycle counter is enabled before calling encoder and decoder functions for each audio frame and stopped right after them. Information is sent to the software together with the encoded data size used to calculate the actual bit-rate.

On the basis of the clock cycle counter information and the audio frame size, the software calculates the Hz used to encode and decode 1 second of audio with the selected Opus configuration. You can compare the selected STM32 system clock with the profiling results and decide if that particular Opus configuration is suitable for your target application, otherwise you can choose a more or less powerful STM32 or change Opus settings to obtain a better audio quality or reduce the resource demand.

Profiling results show other statistics as maximum and minimum values, the actual bit-rate, the original input file size and the encoded file size, etc. Detailed data are saved in a log file for each profiling test.

1.5.3 STCmdP - ST command protocol

This section describes the bit-level message format and the higher level commands of the communication protocol used in the application.

1.5.3.1 *Data format*

The serial protocol defines the following layers:

- Layer 1 - the first lower layer
- Layer 2 - the second upper layer

Table 1. Layer 1 details

Type	Encoded payload	EOF
Bytes	N	1

- **Encoded payload** is the message exchanged between the module and a host through the protocol. The actual payload is encoded by a byte stuffing function to avoid any EOF character in its content.
- **EOF (0xF0)** byte represents the end of the packet. Its value is 0xF0.

Layer 1 encoded payload 1 must be processed by a function which performs the `reverseByteStuffing` to get the payload. The resulting payload is a Layer 2 packet.

Table 2. **Layer 2 details**

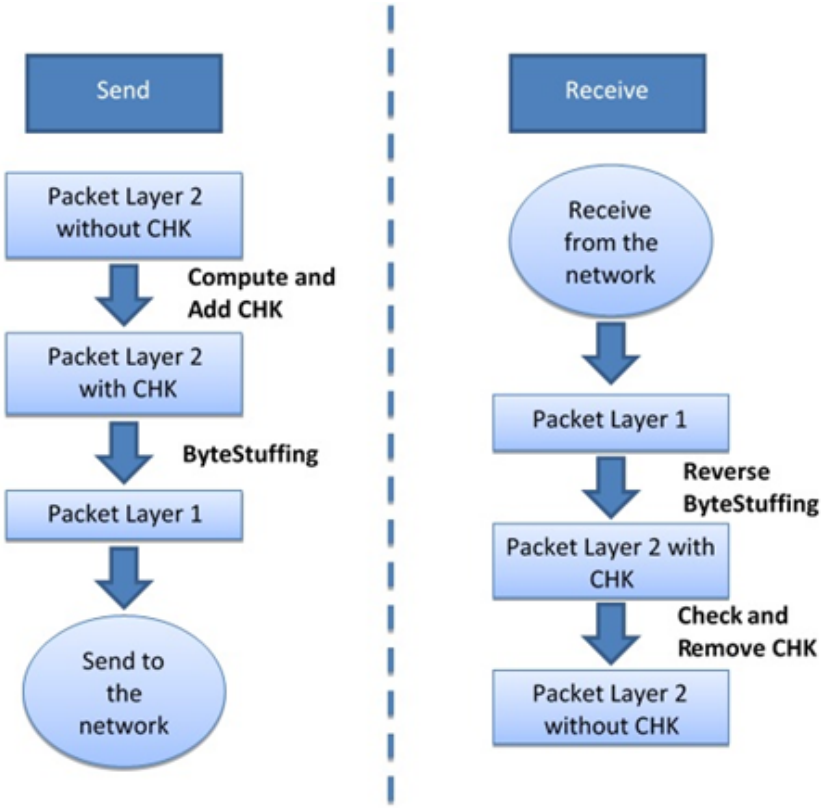
Type	Destination address	Source address	Command (CMD)	Payload	CHK
Bytes	1	1	1	N	1

- **Destination address** is the destination symbolic address which enables unicast messages on a shared bus.
- **Source address** is the source symbolic address.
- **Command** is the code representing the issued command which specifies how to interpret the payload.
- **Payload** data are interpreted with respect to the CMD field.
- **CHK** is the computed sum of all the bytes in the packet which must be zero.

1.5.3.2 *Sending/receiving data packets*

The following figure shows data packet send/receive process. The application of the two functions for `checksum`, `bytestuffing` and their reverses ensures the correct interpretation of the CMD and its payload.

Figure 3. **Send and receive process**



1.5.3.3 *Byte stuffing*

The byte stuffing process transforms a sequence of data bytes that may contain 'illegal' or 'reserved' values into a potentially longer sequence that contains no occurrence of these values.

The EOF character identifies the end of the packet; therefore, the packet must not contain any other occurrence of EOF characters.

To this aim, the following special characters are defined:

- **TMsg_EOF (0xF0)**: the EOF of layer 1 packet
- **TMsg_BS (0xF1)**: the byte stuffing escape character
- **TMsg_BS_EOF (0xF2)**: the substitution for TMsg_EOF

The byte stuffing algorithm, used in sending actions, is defined as described hereafter.

Given a Layer 2 message, for each character, the algorithm:

- substitutes `TMsg_BS` with `TMsg_BS` followed by `TMsg_BS` (it doubles the character)
- 0xF1 becomes 0xF1 0xF1
- substitutes each `TMsg_EOF` with `TMsg_BS` followed by `TMsg_BS_EOF`
- 0xF0 becomes 0xF1 0xF2

As shown, Layer 2 packet length may change in the process.

The reverse byte stuffing algorithm works in the opposite way.

Given a Layer 1 payload, for each character, the algorithm:

- substitutes the expression (`TMsg_BS` followed by `TMsg_BS`) with a single `TMsg_BS`
- 0xF1 0xF1 becomes 0xF1
- substitutes the expression (`TMsg_BS` followed by `TMsg_BS_EOF`) with TMsg_EOF
- 0xF1 0xF2 becomes 0xF0

1.5.3.4 *Standard commands*

STCmdP command protocol contains some standard fundamental commands but can be extended to insert more commands specifically related to the target application.

Table 3. STCmdP standard commands

Command	CMD value	Meaning
CMD_Ping	0x01	This is the standard ping command: the device will reply accordingly.
CMD_Read_PressString	0x02	Requests the presentation string which contains basic device information (name and version).
CMD_Reset	0x0F	Requests a reboot of the device.
CMD_Reply_Add	0x80	This value is added to the value of the CMD field to form the command for the reply (e.g., 0x81 = reply to the PING command).
CMD_NACK	0x03	The request command procedure does not end properly.

This list specifies the CMD field value that can be inserted in the request packet (from the host to the module). The module will reply adding `CMD_Reply_Add` to the CMD value.

Note: The packet length is determined by low level functions using terminator and escape special characters. For this reason, the real length of the packet could not be equivalent to the message length. Moreover, data contained in the Layer 2 payload are serialized before being copied into a packet so that, using a function to deserialize them, data are independent from the architecture (big/little-endian).

In this example, we consider the PING command, how to form the request and its response to Layer 1 and Layer 2.

Request

With the hypothesis of:

- Sender Address = TMsg_EOF = 0xF0
- Destination Address = 0x42

The Layer 2 request packet is as detailed in the following table.

Table 4. Layer 2 request packet

Type	Destination Address	Source Address	Command	CHK
Length (bytes)	1	1	1	1
Value	0x42	0xF0	CMD_Ping = 0x01	205 = 0xCD

After the bytestuffing algorithm application, Layer 1 packet will be:

- 0x42
- TMsg_BS = 0xF1
- TMsg_BS_EOF = 0xF2
- 0x01
- 0xCD
- TMsg_EOF =0xF0

Reply

The packet is received at Layer 1. After the reverse bytestuffing and checksum application, the original packet is parsed (at Layer 2).

Upon a PING request, Layer 2 reply will be as detailed below.

Table 5. Layer 2 reply to a PING request

Type	Destination Address	Source Address	Command	CHK
Length (bytes)	1	1	1	1
Value	0xF0	0x42	CMD_PING + CMD_Reply_Add = 0x81	77 = 0x4D

After the bytestuffing algorithm application, Layer 1 packet will be:

- TMsg_BS = 0xF1
- TMsg_BS_EOF = 0xF2
- 0x42
- 0x81
- 0x4D
- TMsg_EOF =0xF0

1.5.3.5 *Custom commands*

Table 6. OpusProfilingTool application specific commands

Command	CMD Value	Meaning
CMD_Init_Opus_Encoder	0x46	Requests Opus encoder initialization; it contains all the necessary configuration settings
CMD_ Init_Opus_Decoder	0x47	Requests Opus decoder initialization, it contains all the necessary configuration settings
CMD_ Reset_Opus	0x48	Requests Opus encoder and decoder reset
CMD_EncDec_NoFile_Request	0x49	Requests to encode and decode received audio data; the reply will contain profiling data but not encoded and decoded data
CMD_ EncDec_File_Request	0x4A	Requests to encode and decode received audio data; the reply will contain profiling data together with encoded and decoded data

1.6 Opus profiling tool software control

X-CUBE-OPUS package contains a Python application that must be used together with the firmware example running on a supported STM32 Nucleo board to configure and profile Opus encoder and decoder.

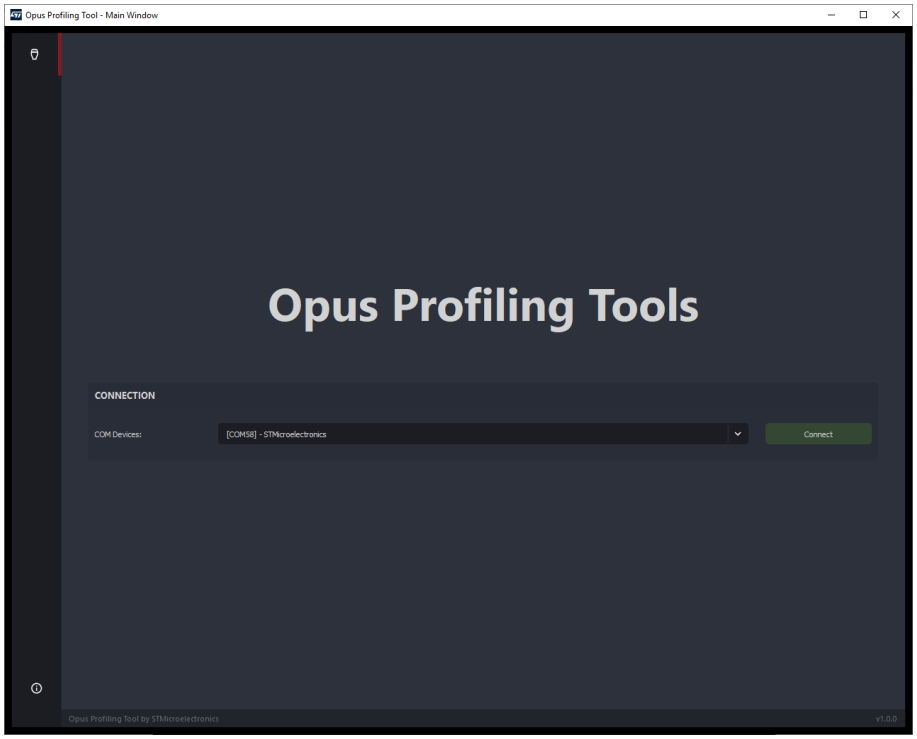
The *Utilities\PC_software\OPT_SW_Control* folder contains all the material needed to run and use the software control:

- an executable file
- a folder containing audio samples that can be used to run profiling
- a quick start guide

1.6.1 Connection

After flashing a supported STM32 Nucleo board with the relevant firmware, connect it to the PC through ST-LINK. As soon as the software is launched, a list of Virtual COM ports is shown: choose the correct one, after checking the Windows Device Manager, and press the [**Connect**] button to enable the profiling window.

Figure 4. Opus Profiling Tools connection



You can go back to the connection page by clicking on the relevant icon and disconnect the board by pressing the button, or simply unplugging the board any time and the software will automatically go back to the connection page.

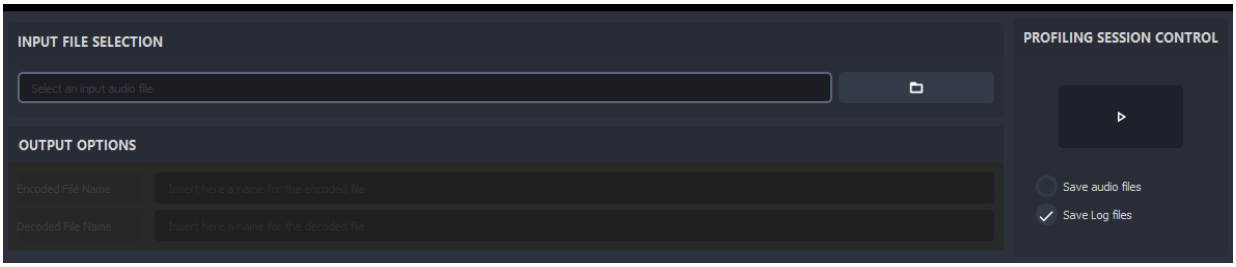
Press the connect button to open the main page, which could be divided in three sections: input and output, encoder and decoder parameters, Opus memory footprint.

1.6.2 Input and output

In the main page first section you can choose an audio file to be used for the profiling test among the supported audio formats: wav, mp3, ogg, flv, raw. If you choose a raw file (only PCM data), you need to specify the audio sampling frequency and number of channels in the encoder parameters.

If a correct input file is chosen, the start button becomes green, otherwise an error message is shown.

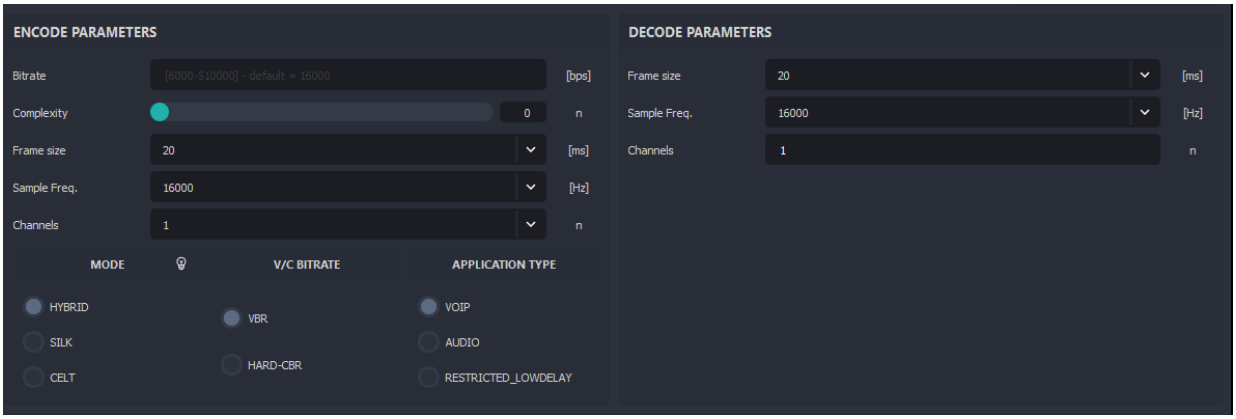
Figure 5. Input file selection and output options



1.6.3 Encode/Decode parameters

The main page second section is dedicated to encode and decode parameters. There are several Opus settings to choose from. When the start button is pressed, initialization commands are sent to the board (refer to Section 1.5.1 Opus for further details on parameters).

Figure 6. Encode/Decode parameters



The sampling frequency and number of channels are automatically selected if a non-raw input file is chosen, since they are extracted directly from the audio file header. However, you can decide to change the decoder audio sampling frequency and Opus will automatically up-sample or down-sample the final output.

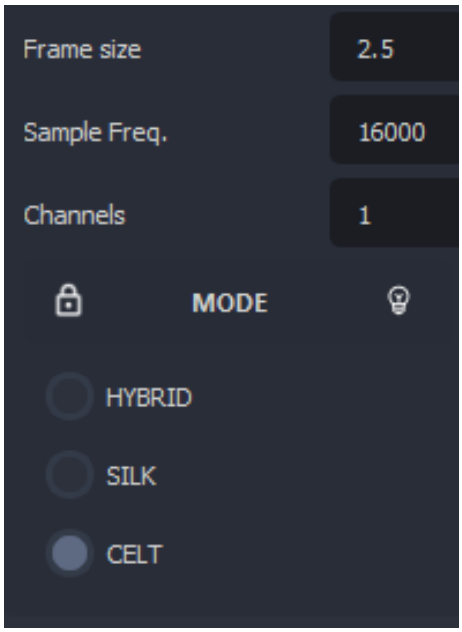
The frame size must be the same for encoder and decoder, so it automatically switches when you choose one of them.

Selecting the [**Mode**] radio button you can force Opus to use one of the three possible algorithms mode: Hybrid, Silk and Celt.

Note: It is not always possible to force an algorithm mode: depending on the particular configuration chosen, the radio button might lock (for example, if you select frame size equal to 2.5 ms only Celt is possible), or Opus might switch from one algorithm to another depending on the audio data.

A plot in the result window shows the actual algorithm used; moreover, in the log file the information is reported for each frame.

Figure 7. Mode



1.6.4 Opus memory footprint

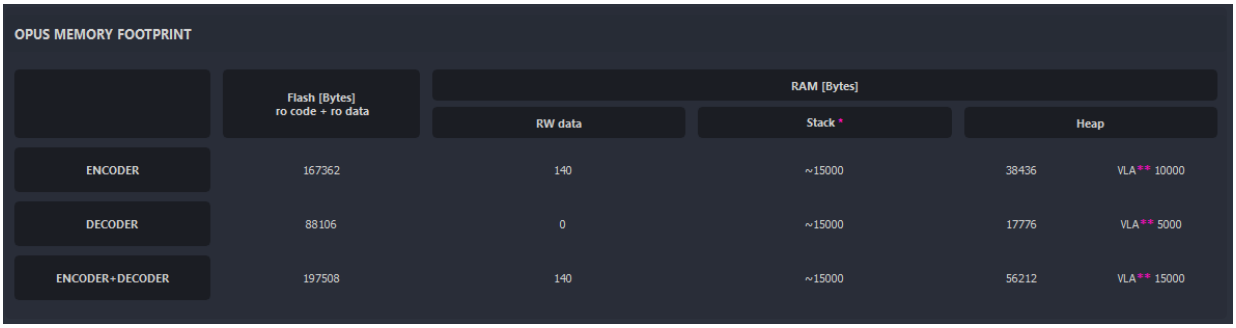
The main page last section shows an estimation of the memory used by the MCU based on the selected configuration. It is divided in Encoder, Decoder and Encoder+Decoder if you use both.

In the first column the amount of flash used for encoder, decoder or both is shown.

In the second part the RAM usage is shown, divided in read/write data, stack and heap.

Note: The stack value is just an estimation and can vary depending on particular configurations. The heap size is divided between a specific value, allocated by Opus during encoder and decoder initialization (it changes only when the number of channels change), and an estimated value for the variable length array allocated by Opus at run-time. This last value can change depending on the configuration.

Figure 8. Memory footprint



1.6.5 Profiling processing and results

After selecting all the desired settings and pressing the play button, the profiling results window pops up.

It shows all the data received from the STM32 Nucleo board in real-time. A summary of the Opus parameters chosen is displayed on the top left corner. At the page bottom, a progress bar shows the percentage of audio data already profiled.

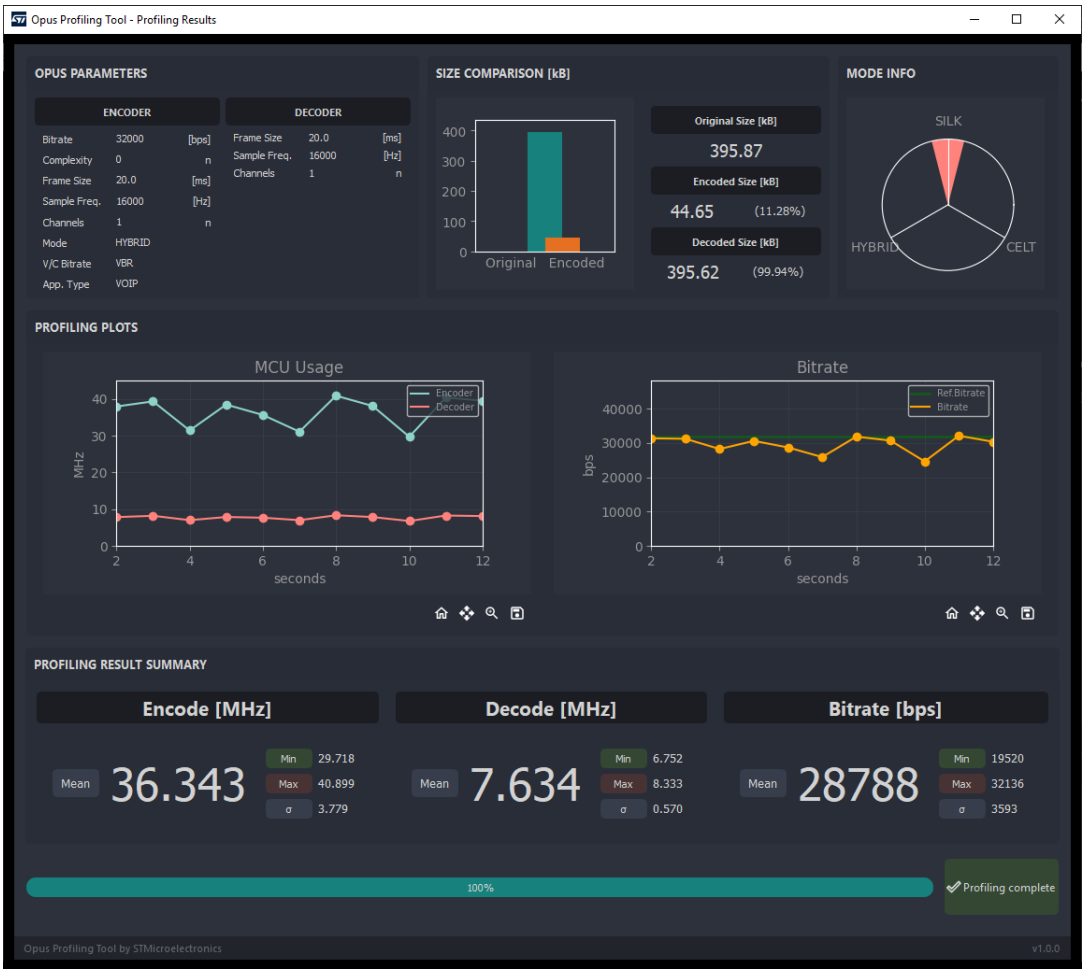
The *Size comparison* section shows the original and the encoded size. The decoded size can vary a bit with respect to the original: the difference is less than an Opus frame size and it cannot be analyzed.

On the top right corner the *Mode info* plot displays which Opus algorithm is used during the profiling. The *Profiling plots* below shows the MHz requested for each second of audio analyzed for both encoder and decoder on the left and the actual bit-rate on the right.

Remember: The actual bit-rate can vary if you have selected variable bit-rate (VBR).

Lastly, a profiling result summary lists some statistics such as average, max., min., standard deviation for encoder and decoder MCU usage and for bit-rate.

Figure 9. **Profiling results**



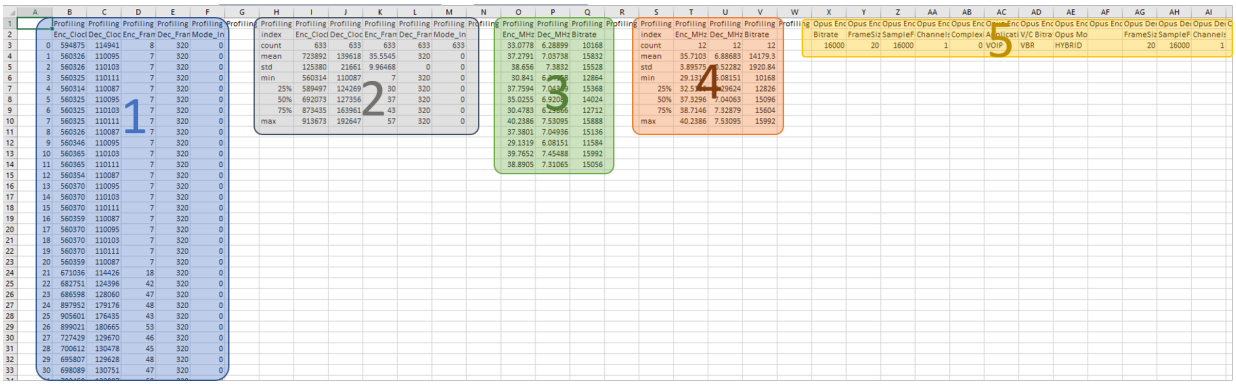
1.6.6 Log and audio output file

All the results and data displayed by the profiling result window can be also saved in output file by selecting the relevant option in the main window.

Audio and log file are saved in the same path of the audio input file, in a dedicated folder. The *Audio* folder contains the encoded audio file and the result of encode and decode so that you can compare audio quality with respect to the original file. In the *LOG* folder a csv file is saved, divided in the following sections:

1. Profiling data for each audio frame
2. Statistics related to the frame by frame profiling in section 1
3. Encoder and decoder MCU usage and bit-rate for each second of audio analyzed
4. Statistics related to profiling data for each second in section 3
5. Summary of Opus settings chosen for the current profiling

Figure 10. **Profiling results**



2 System setup guide

2.1 Hardware description

2.1.1 STM32 Nucleo

STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

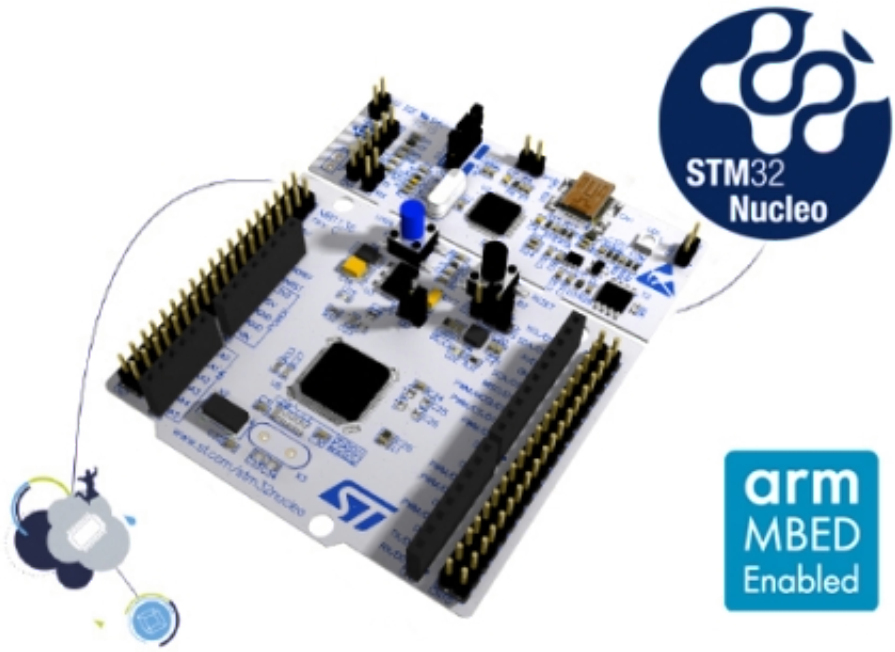
The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples for different IDEs (IAR EWARM, Keil MDK-ARM, STM32CubeIDE, mbed and GCC/ LLVM).

All STM32 Nucleo users have free access to the mbed online resources (compiler, C/C++ SDK and developer community) at www.mbed.org to easily build complete applications.

Figure 11. STM32 Nucleo board



Information regarding the STM32 Nucleo board is available at www.st.com/stm32nucleo

2.1.2 P-NUCLEO-WB55

The P-NUCLEO-WB55 pack is a multi-protocol wireless and ultra-low-power device embedding a powerful and ultra-low-power radio compliant with the Bluetooth® Low Energy (BLE) SIG specification v5.0 and with IEEE 802.15.4-2011.

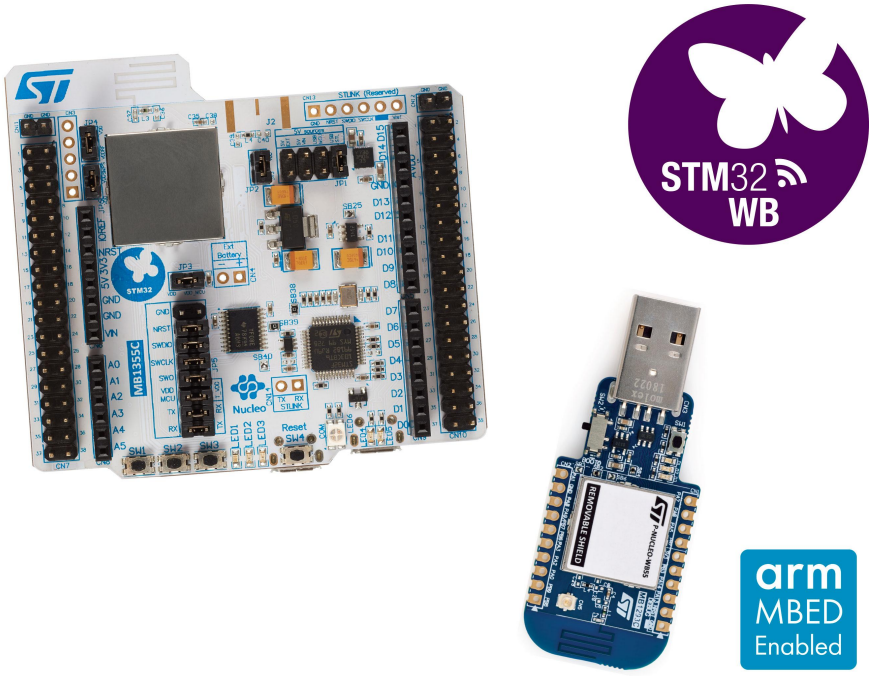
The pack consists of a Nucleo-68 development board and a USB dongle.

It features:

- STM32WB microcontroller in a VFQFPN68 package
- 2.4 GHz RF transceiver supporting Bluetooth® specification v5.0 and IEEE 802.15.4-2011 PHY and MAC

- Dedicated Arm® 32-bit Cortex® M0+ CPU for real-time Radio layer
- Three user LEDs
- Three user buttons and one reset button
- Board connector: USB user with Micro-B
- Arduino™ Uno V3 connector
- ST morpho connectors
- Integrated PCB antenna or footprint for SMA connector
- Flexible power-supply options: ST-LINK USB VBUS or external sources
- On-board socket for CR2032 battery
- On-board ST-LINK/V2-1 debugger/programmer with USB re- enumeration capability: mass storage, virtual COM port and debug port
- Comprehensive free software libraries and examples available with the STM32Cube package
- Support of a wide choice of Integrated Development Environments (IDEs), including IAR™, Keil®, GCC-based IDEs, Arm® Mbed™

Figure 12. P-NUCLEO-WB55 development pack



2.2 Software description

The following software components are needed in order to setup the suitable development environment to evaluate and profile applications for STM32 based on the Opus audio codec:

- X-CUBE-OPUS: an expansion for STM32Cube dedicated to Opus audio codec. The X-CUBE-OPUS firmware and related documentation is available on www.st.com.
- Development tool-chain and Compiler. The STM32Cube expansion software supports the three following environments:
  - IAR Embedded Workbench for ARM® (EWARM) toolchain + ST-LINK
  - Keil Microcontroller Development Kit (MDK-ARM) toolchain + ST-LINK
  - STM32CubeIDE for STM32 (SW4STM32) + ST-LINK

2.3 PC utility

The *OPT_SW_Control* utility for the host device has following minimum requirements:

- a PC with Intel or AMD processor running Windows 7 or Windows 10
- At least 50 MBs of RAM
- 1 X USB ports
- 50 MB of hard disk space

The *OPT_SW_Control* can be used to interact with and get data from the Opus profiling firmware running on the STM32 Nucleo development board.

The utility can be launched by clicking twice on the OPT_SW_Control.exe file, located in the “Utilities\PC_software\OPT_SW_Control” folder.

2.4 Hardware setup

The following hardware components are needed:

1. One STM32 Nucleo development platform (suggested order code: NUCLEO-F413ZH, NUCLEO-H743ZI, NUCLEO-F746ZG, NUCLEO-L552ZE-Q or P-NUCLEO-WB55)
2. One USB type A to Micro-B USB cable to connect the STM32 Nucleo to the PC

2.4.1 P-NUCLEO-WB55 setup

When using P-NUCLEO-WB55 to test the X-CUBE-OPUS software, an additional operation should be done before starting.

To support the various Opus configurations for a complete profiling tool, the firmware requires a great amount of RAM allocation; the P-NUCLEO-WB55 requires all the available RAM as well as the RAM shared by the two cores (Cortex-M4 for the application and Cortex-M0+ for RF). To avoid issues, it is suggested to delete the RF stack (if it is flashed).

For further details, refer to steps 1, 2 and 3 of the guide available in the STM32CubeWB package (how_to_program_wireless_stacks.txt and the relevant Release Notes).

Revision history

Table 7. **Document revision history**

Date	Version	Changes
11-Dec-2020	1	Initial release.

Contents

List of tables

List of figures