

Implementing OPUS Voice Code for TM4C129x Device

Amit Ashara

ABSTRACT

Opus is an open source, royalty-free audio compression format developed by Xiph and standardized by the IETF. It utilizes lossy compression which is designed to efficiently code audio with a low latency making it suitable for real time communication. Opus replaces both the Vorbis and Speex codecs, and is intended for storage and streaming applications. Its low complexity allows it to be run efficiently on TM4C129x series microcontroller from Texas Instruments. This document describes how to use the Code Composer Studio™ v6.1.1 software to build Opus and the examples that can be used to exercise compression and decompression functions provided.

Project collateral and source code mentioned in this document can be downloaded from the following URL: <http://www.ti.com/lit/zip/spma076>.

NOTE: This document applies to TM4C129x series microcontrollers. All screen captures reflect the TM4C129x device and Code Composer Studio v6.1.1 IDE.

Contents

1	Introduction	2
2	System Details	2
3	Example Code Description	4
4	Executing OPUS Example Project	14
5	Performance Data on OPUS Encoder	20
6	Conclusion	22
7	References	22

List of Figures

1	Custom OPX Audio Format	3
2	Import the Examples	5
3	Import the Examples	6
4	Compiling opuslib	7
5	Edit Path Variables	8
6	opus_enc_dec Data Flow	9
7	Compiling opus_enc_dec example	10
8	opus_playaudio_opx Data Flow	11
9	Compiling opus_playaudio_opx Example	12
10	opus_playaudio_ogg Data Flow	13
11	Compiling opus_playaudio_ogg Example	14
12	Application Encoding a Wav File to OPX Format	15
13	Application Prints Statistics for Compression Operation	15
14	Application Decoding OPX File to a Wave File	16
15	Application Prints Statistics for the Decompression	16

Code Composer Studio, TivaWare are trademarks of Texas Instruments.
 ARM, Cortex are registered trademarks of ARM Limited.
 All other trademarks are the property of their respective owners.

16	Application opus_playaudio_opx as Seen on the LCD Panel	17
17	Application opus_playaudio_opx Running the Decoder.....	18
18	Application opus_playaudio_ogg as Seen on the LCD Panel.....	19
19	Application opus_playaudio_ogg Running the Decoder	20

List of Tables

1	Opus Encoder Performance for 16-Bit Audio Data Sampled at 8 KHz	21
2	Opus Encoder Performance for 16-Bit Audio Data Sampled at 16 KHz	21

1 Introduction

Many microcontroller-based applications require audio to be either recorded or played back. Using raw audio formats require external storage as the size of the files are large and cannot be stored in the on-chip flash. Other audio formats may require special licensing agreements that require fees to be paid increasing the cost of the system development. The open source structure allows the development of applications without having to spend on licensing cost. Also with the wider adoption of the codec, it makes a good case for microcontroller applications to utilize the benefits of the codec especially in IoT applications.

The main features of the Opus code are listed below:

- Bitrates from 6Kbits/s to 510Kbits/s
- Five sampling rates from 8 KHz to 48 KHz: 8 KHz, 16 KHz, 24 KHz, 32 KHz and 48 KHz
- Frame sizes from 2.5 ms to 60 ms
- Support for both constant bitrate (CBR) and variable bitrate (VBR)
- Audio bandwidth from narrowband to full band
- Support both spec and music, mono and stereo
- Good loss robustness and packet loss concealment (PLC)
- Floating- and fixed-point implementations

The TM4C129x series microcontroller features:

- ARM® Cortex® M4F processor core at 120 MHz
- 1 Mbyte of on-chip flash and 256 Kbyte of on chip SRAM
- Integrated security modules like AES and DES and SHA hash engine
- Integrated 10/100 Ethernet MAC and PHY for network communication and IoT solutions
- Single 32-channel uDMA for data transfer
- Two 12-bit ADC modules each with a maximum of 2MSPS
- 8 16/32-bit configurable timer block
- USB2.0 OTG/Host/device with ULPI interface and Link Power Management (LPM) support

Even though there is no native support for SD card or Audio Interface, TM4C129x with its large on chip flash and SRAM, CPU core at 120 MHz can efficiently handle the Opus codec for play back on USB interface. Optionally, as described in this application report, the SD card can be read by the microcontroller over SPI interface using legacy mode and a timer in PWM mode can be used to playback audio with sufficient clarity for low and mid end applications and systems, reducing the requirement for more complex systems.

2 System Details

To successfully integrate the Opus Codec into an application, the system must meet the following requirements.

2.1 Hardware Requirements

The application example uses the DK-TM4C129X EVM from Texas Instruments. The EVM has an on-board SD card slot for reading files and buzzer for audio playback. To run the examples as given in the application note the user must install jumpers for the following headers.

- For the SD card interface, install all headers on J7 connector except one marked as FLS_CS
- For the audio interface, install the two headers (J21 and J22) marked as SPEAKER CON

2.2 Software Requirements

All examples built as part of the application report assume that a linear PCM stream is used with sampling rates from 8 KHz up to 16 KHz and a single channel.

1. Download and install TivaWare™ 2.1.2.111.
2. Create a directory “examples\OPUS” in “D:\ti”.
3. Download the source code from the opus website, <https://opus-codec.org/downloads/>, and unzip it. The version of opus source code used at the time of this document is libopus 1.1.2.
4. Download the project collateral and unzip it.
5. Install CCS v6.1.11 with ARM compiler tool chain version 5.2.6.

2.3 Audio Format

Opus prescribes the OggS file container for storing files or data streams encoded using the opus codec. The playback example along with this application report support OggS-Opus as well as a custom format. Information on the OggS-Opus is available online. For references, see Section 7. The custom format is referenced in this document as the OPX format. The OPX format has been developed to simplify the code for file processing during both compress and decompress operation of the codec. The details for OPX are shown in the Figure 1.

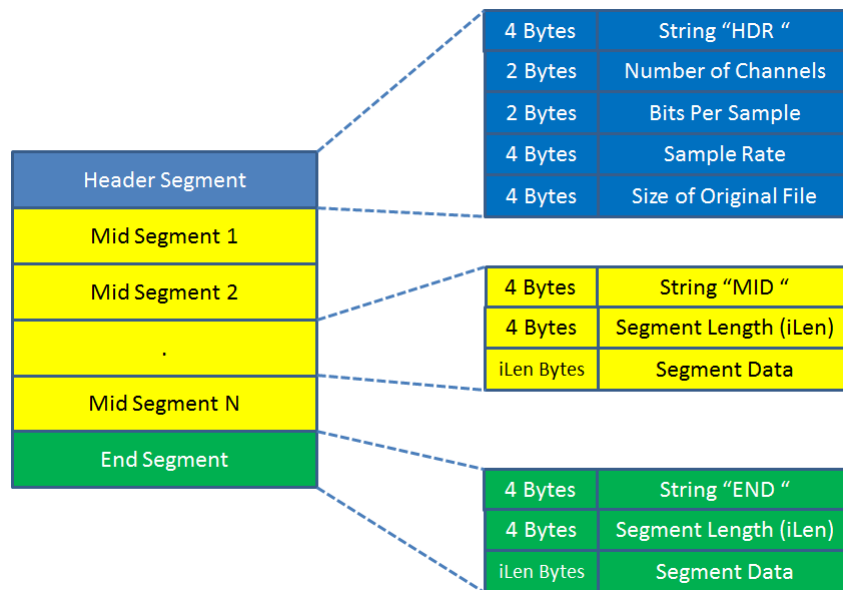


Figure 1. Custom OPX Audio Format

The OPX audio format contains three distinct sections referred to as segments.

2.3.1 Header Segment

The header segment is an information only segment and does not have any audio data. It is always 16 bytes long and has five fields:

- The first 4 bytes contain the string "HDR " (Hex string of 0x48 0x44 0x52 0x00), which is used to identify the validity of the file.
- The next 2 bytes contain the number of channels. This can have the value "0x0001" for mono or "0x0002" for stereo.
- The next 2 bytes contain the bits per sample in the original bit stream. This can have the value "0x0008" for 8 bit data or "0x0010" for 16 bit data.
- The next 4 bytes contain the sampling rate of the original bit stream in Hz.
- The last 4 bytes contain the size of the data payload in the original bit stream.

2.3.2 OPX Mid Segment

The mid segment is a data only segment and must be 0 or more. Each mid segment contains data equivalent to the frame size used during encode process.

- The first 4 bytes contains the string "MID " (Hex string of 0x4D 0x69 0x64 0x00), which is used to identify that it is a data payload and that it is not the last payload segment.
- The next 4 bytes, in 32-bit unsigned type, contain the size of the segment data in bytes called "iLen".
- The last field is the actual data and is "iLen" bytes long

2.3.3 OPX End Segment

The end segment is a data only segment and must be exactly one. The end segment contains data equivalent to the frame size used during encode process.

- The first 4 bytes contains the string "END " (Hex string of 0x45 0x6E0x64 0x00), which is used to identify that it is a data payload and it is the last payload segment.
- The next 4 bytes, in 32-bit unsigned type, contains the size of the segment data in bytes called "iLen".
- The last field is the actual data and is "iLen" bytes long

3 Example Code Description

The CCS project examples are part of the project collateral included with this application report. This project collateral includes four CCS projects:

- **opuslib**: This project is used to compile the Opus source code to create a pre-compiled library that is used in the application examples. The output of this project is a static library "opuslib.lib" that must be used during the linker phase of the other application examples.
- **opus_enc_dec**: This project is a serial console-based application for an embedded device that is used to read a wave file from the sd-card, compress the wave file to an OPX file and store the OPX file on the sd-card. It performs the reverse steps to read an OPX file from the sd-card, convert it to a wave file and store it on the sd-card.
- **opus_playaudio_opx**: This project is a graphics application for an embedded device that reads an OPX file from the sd-card, decompresses the data to play back on the audio buzzer on the DK-TM4C129X EVM. To playback OPX audio file, run the opus_enc_dec application to convert a mono format wav file to OPX file format.
- **opus_playaudio_ogg**: This project is a graphics application for an embedded device that reads an Oggs-Opus file from the sd-card and decompresses the data to play back on the audio buzzer on the DK-TM4C129X EVM. To playback the opus audio file, you can download opus files in mono format from the web.

3.1 Importing the Examples

Before building the opus library in Code Composer Studio, a project for the library must be created. The following steps describe how to create and build the project:

1. Launch CCS v6.1.1 and select an empty workspace.
2. Select File → Import. The *Import* window will be displayed.
3. Click *Code Composer Studio* to expand then click *CCS Projects*. Click Next (see [Figure 2](#)).

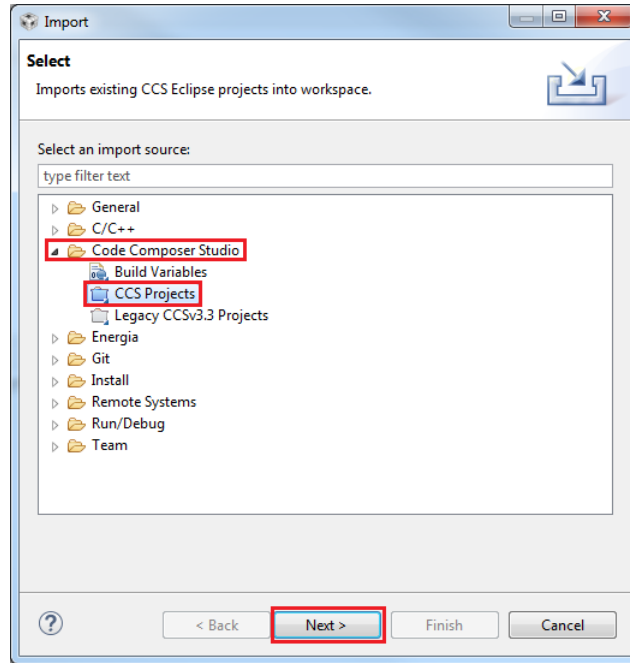


Figure 2. Import the Examples

- Click the *Browse* button in front of *Select search-directory* option and navigate to the directory where the project collateral has been extracted. Select all the projects that are listed in the *Discovered projects* pane. Make sure that the check-boxes in front of *Automatically import referenced projects found in same search-directory* and *Copy projects into workspace* are checked (see [Figure 3](#)). Now click on *Finish* button. This will import the examples into CCS.

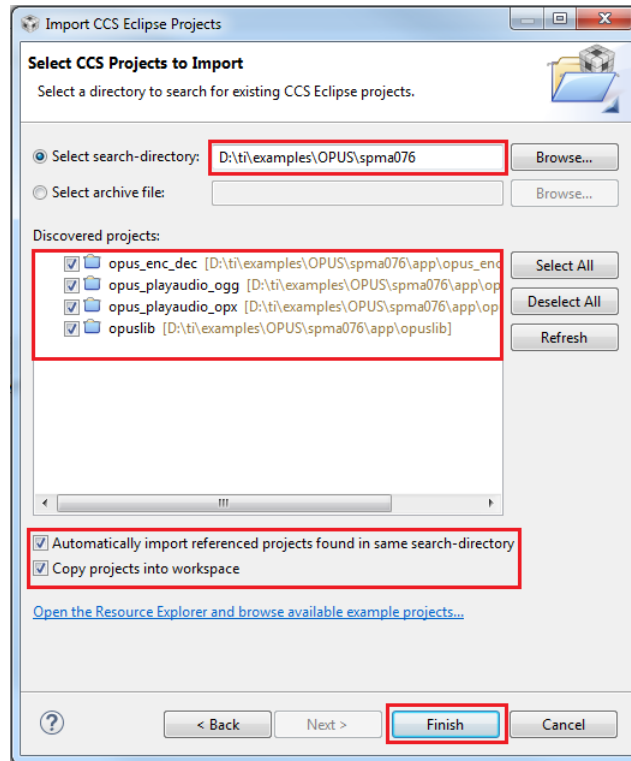


Figure 3. Import the Examples

3.2 Compiling opuslib

Since all the application examples refer to the opuslib project, the opuslib must be compiled first. Right-click the opuslib project in Project Explorer and click *Build Project*. If building the library for the first time, it may take a few minutes. After the compilation is successful, the CCS console must display the following message (see [Figure 4](#)).

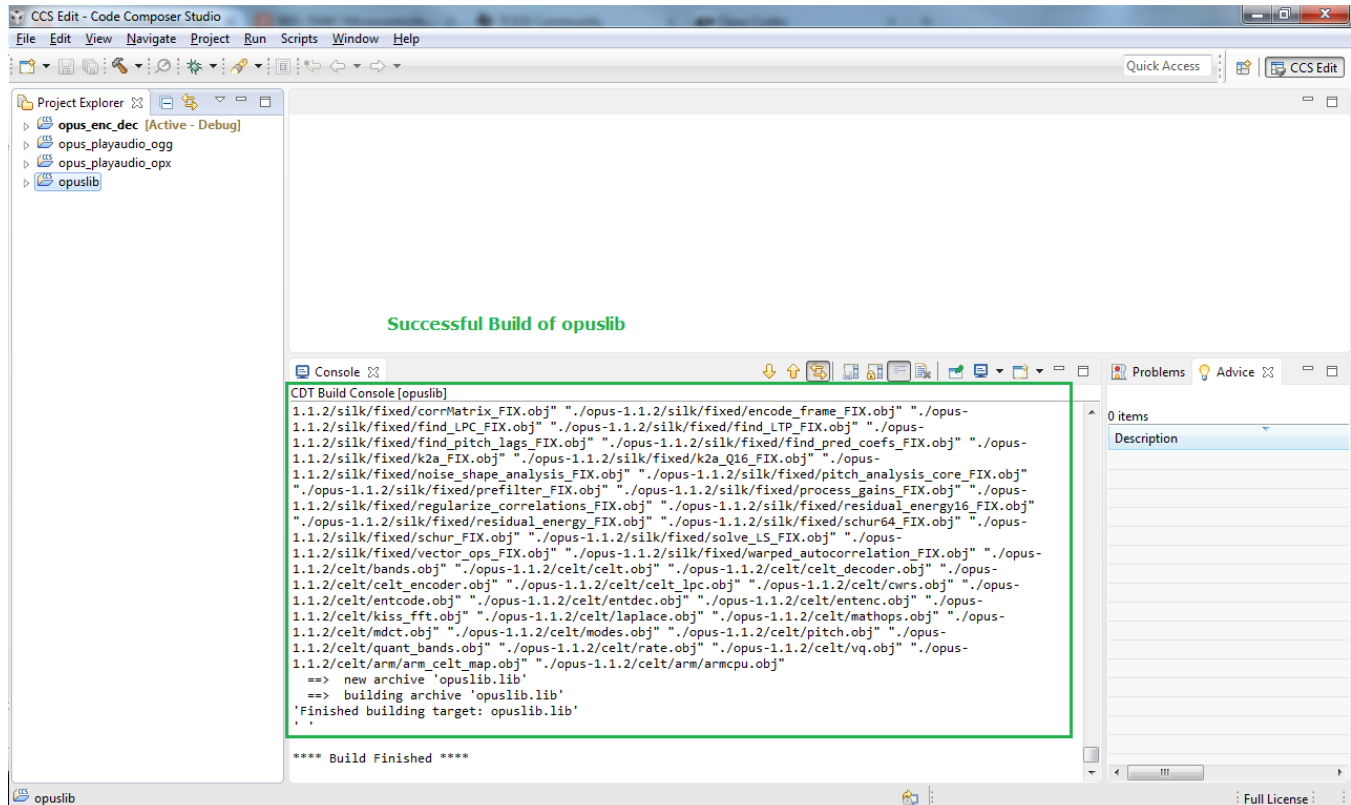


Figure 4. Compiling opuslib

3.3 Compiling the Application Examples

If the required software has been extracted or installed (as provided in Section 2.2) to a path not the same as specified by the variables, then you must first modify the value of these variables in the respective projects (see Figure 5). The following steps show how to update the three variables (SW_ROOT, OPUS_ROOT and SPMA076_ROOT).

1. Right click the project, click *Show Build Settings...*
2. Select *Build*, click on the *Variables* tab, select the variable to modify and click on the *Edit...* button

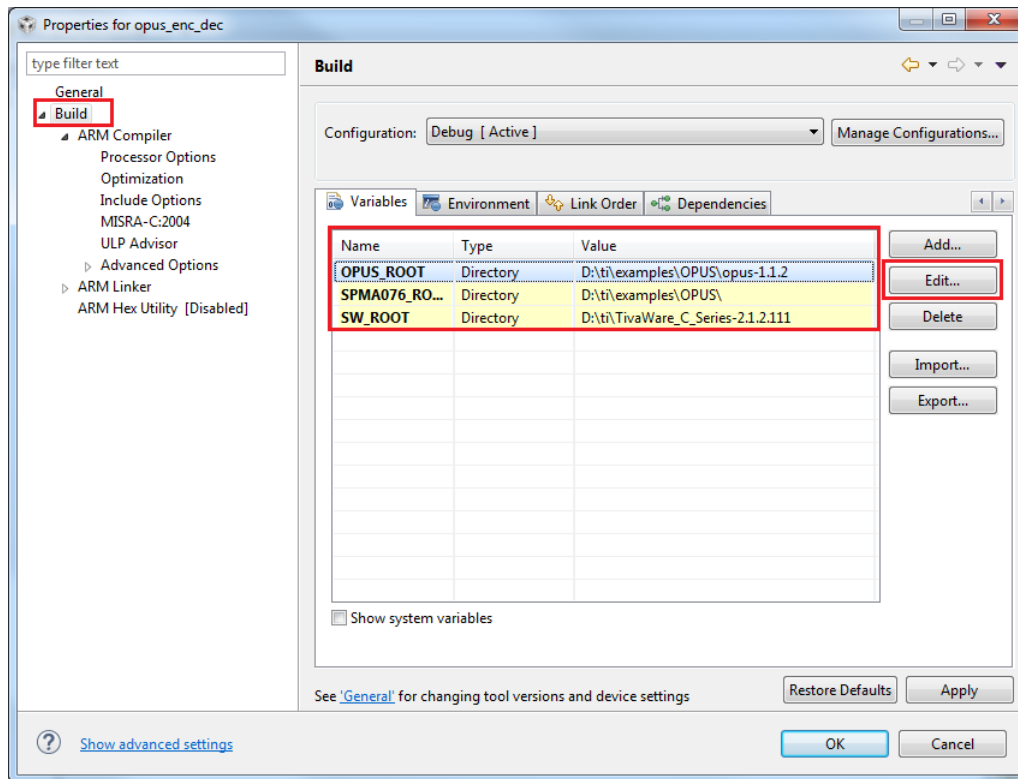


Figure 5. Edit Path Variables

3.3.1 Compiling Opus Encode and Decode Example

The opus_enc_dec project contains the example for demonstrating the opus encoder and decoder with a serial console (TeraTerm or PuTTY) application. In this application example (see Figure 6), when the encode command line option is entered on a serial console, the TM4C129XNCZAD device reads wave file from a SD Card and encodes with the Opus Encoder and stores it back in the SD Card as a OPX file. Similarly, when the decode command line option is entered on a serial console, the TM4C129XNCZAD device reads the OPX file from the SD Card and decodes it with the Opus Decoder and stores the data back in the SD Card as a wave file.

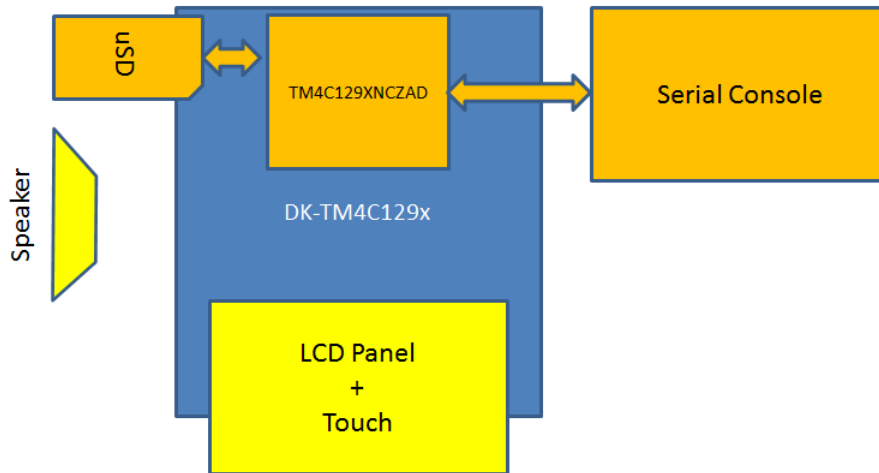


Figure 6. opus_enc_dec Data Flow

To build the opus_enc_dec application, right-click the opus_enc_dec project in Project Explorer and click *Build Project*. After the compilation is successful, the CCS console must display the following message (see Figure 7).

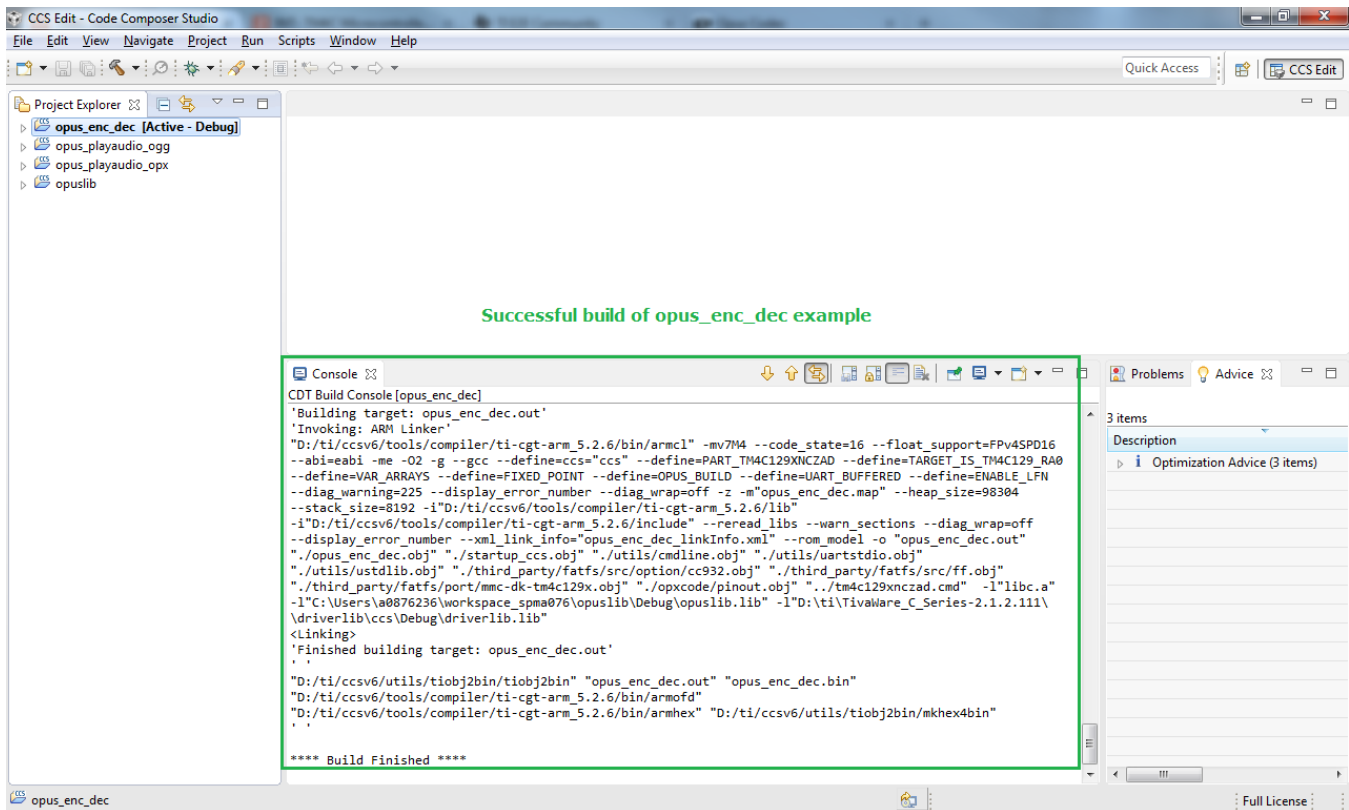


Figure 7. Compiling opus_enc_dec example

3.3.2 Compiling Playback Example for OPX

The `opus_playaudio_opx` project contains the example for demonstrating the opus decoder and playback with a GUI on the LCD panel of the DK-TM4C129X EVM. In this application example (see [Figure 8](#)), the TM4C129XNCZAD reads the SD Card and displays its content on the LCD panel of the DK-TM4C129X. Select an OPX file and playback the audio by using the on-board speaker. The LCD panel provides the touch interface as well for playing, pausing or stopping the audio stream, or selecting another OPX file for playback.

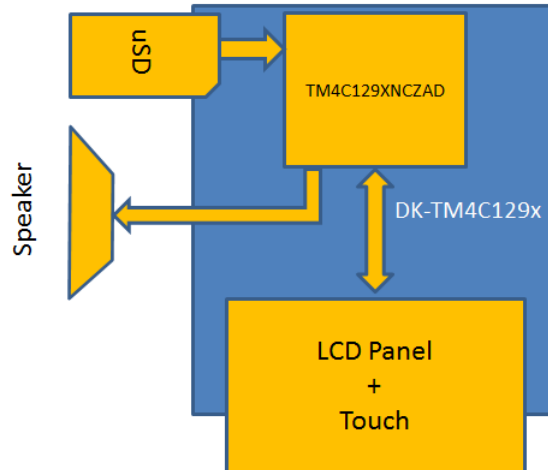


Figure 8. `opus_playaudio_opx` Data Flow

To build the `opus_playaudio_opx` application, right-click the `opus_playaudio_opx` project in Project Explorer and click *Build Project*. After the compilation is successful, the CCS console must display the following message (see [Figure 9](#)).

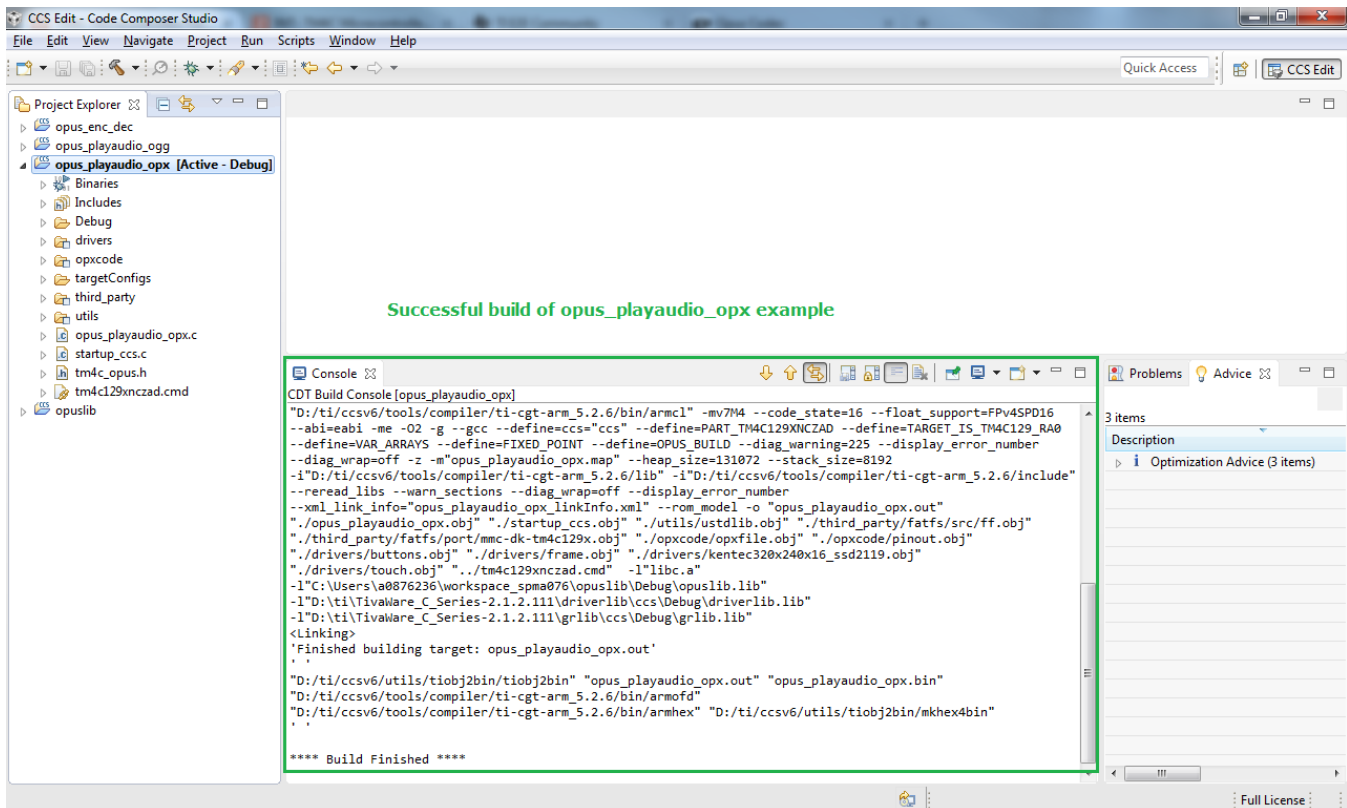


Figure 9. Compiling `opus_playaudio_opx` Example

3.3.3 Compiling Playback Example for OggS-Opus

The `opus_playaudio_ogg` project contains the example for demonstrating the opus decoder and playback with a GUI on the LCD panel of the DK-TM4C129X EVM. In this application example (see [Figure 10](#)), the TM4C129XNCZAD reads the SD Card and displays its content on the LCD panel of the DK-TM4C129X. Select an OggS-Opus file and playback the audio by using the on-board speaker. The LCD panel provides the touch interface as well for playing, pausing or stopping the audio stream, or selecting another OggS-Opus file for playback.

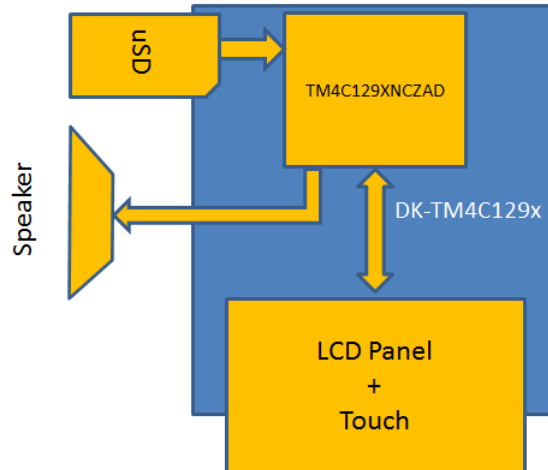


Figure 10. `opus_playaudio_ogg` Data Flow

To build the opus_playaudio_ogg application, right-click the opus_playaudio_ogg project in Project Explorer and click *Build Project*. After the compilation is successful, the CCS console must display the following message (see [Figure 11](#)).

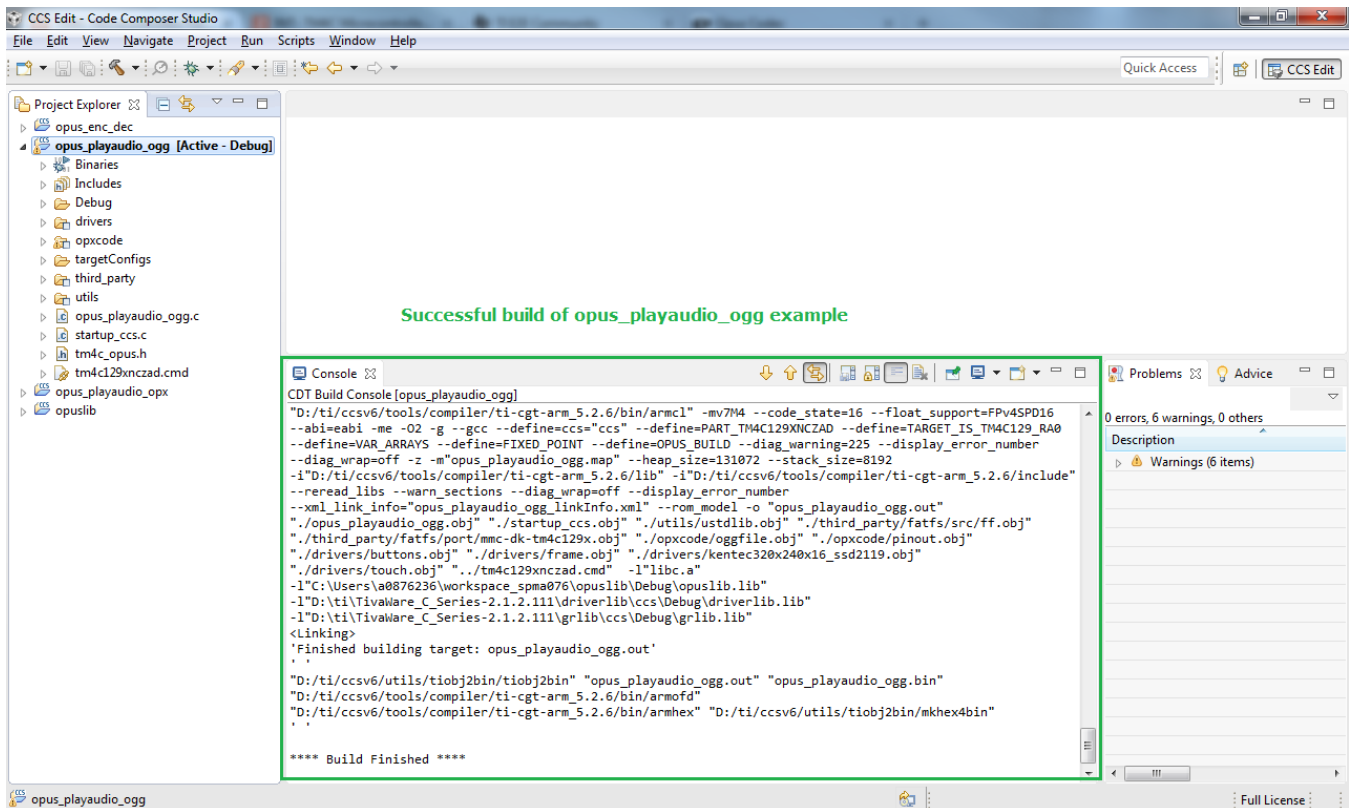


Figure 11. Compiling opus_playaudio_ogg Example

4 Executing OPUS Example Project

To execute the application examples on the DK-TM4C129X, click on the Debug Icon on the top panel of CCS.

4.1 OPUS Encode and Decode

Once the code has been flashed to the TM4C129XNCZAD, run the application example as shown below:

1. Launch a serial console application like TeraTerm (Baud rate of 115200 bps and 8N1 format) and press the Resume button (or F8) in Code Composer Studio to cause the program to start execute. The TeraTerm will show the welcome message. Use the following command:

```

/> enc <WAVFILE> <OPXFILE>
    
```

to encode a wave file to OPX file (see [Figure 12](#)). Once the encoding is complete, the statistics for compression will print (see [Figure 13](#)).

```

COM47:115200baud - Tera Term VT
File Edit Setup Control Window Help
OPUS Encode and Decode Example...
/> ls
D---- 2016/01/21 11:04      0 test
---A 2016/04/25 11:34 25841760 s_16m_48k.wav
---A 2016/04/25 13:59 2153522 s_8m_8k.wav
D---- 2016/04/25 16:06      0 old

  2 File(s), 27995282 bytes total
  2 Dir(s), 738413778K bytes free

/> enc s_8m_8k.wav s_8m_8k.opx
Encoding 1 channel 8 bits 8000 Hz WAV file
OPUS Encoder Completion: 016
    
```

Figure 12. Application Encoding a Wav File to OPX Format

```

COM47:115200baud - Tera Term VT
File Edit Setup Control Window Help
OPUS Encode and Decode Example...
/> ls
D---- 2016/01/21 11:04      0 test
---A 2016/04/25 11:34 25841760 s_16m_48k.wav
---A 2016/04/25 13:59 2153522 s_8m_8k.wav
D---- 2016/04/25 16:06      0 old

  2 File(s), 27995282 bytes total
  2 Dir(s), 738413778K bytes free

/> enc s_8m_8k.wav s_8m_8k.opx
Encoding 1 channel 8 bits 8000 Hz WAV file
OPUS Encoder Completion: 100
*****STATISTICS*****
Raw WAV Bytes      = 2153478
Encoded OPX Bytes  = 507655
Compression Factor = 04.24

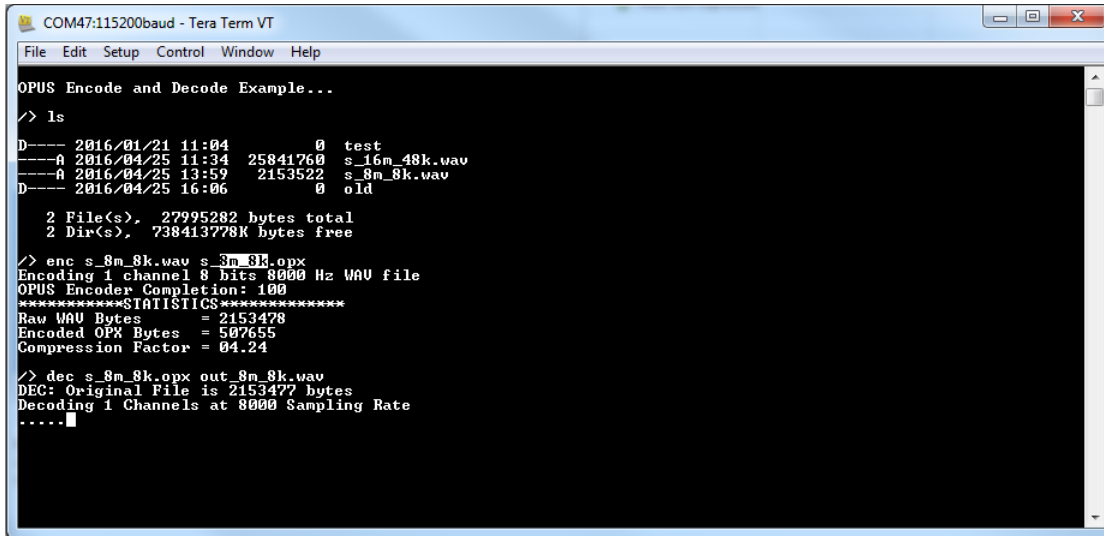
/> □
    
```

Figure 13. Application Prints Statistics for Compression Operation

2. Use the following command:

```
 /> dec <OPXFILE> <WAVFILE>
```

to decode the OPX file to a wave file (see [Figure 14](#)). Once the decompression is complete, it will print the statistics on conversion (see [Figure 15](#)).



```

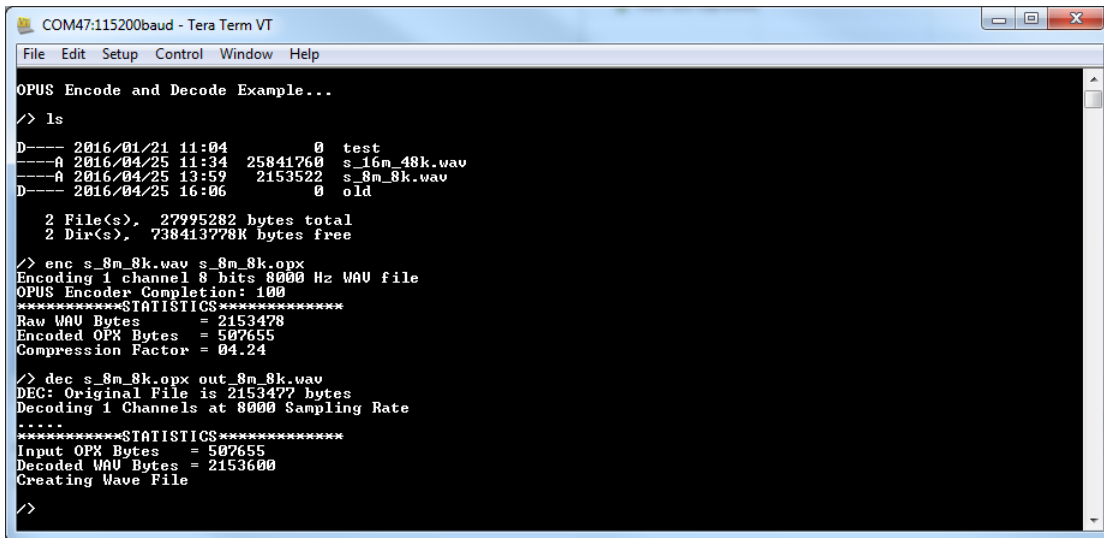
COM47:115200baud - Tera Term VT
File Edit Setup Control Window Help
OPUS Encode and Decode Example...
/> ls
D---- 2016/01/21 11:04      0 test
--A  2016/04/25 11:34 25841760 s_16m_48k.wav
--A  2016/04/25 13:59 2153522  s_8m_8k.wav
D---- 2016/04/25 16:06      0 old

  2 File(s), 27995282 bytes total
  2 Dir(s), 738413778K bytes free

/> enc s_8m_8k.wav s_8m_8k.opx
Encoding 1 channel 8 bits 8000 Hz WAU file
OPUS Encoder Completion: 100
*****STATISTICS*****
Raw WAU Bytes      = 2153478
Encoded OPX Bytes  = 507655
Compression Factor = 04.24

/> dec s_8m_8k.opx out_8m_8k.wav
DEC: Original File is 2153477 bytes
Decoding 1 Channels at 8000 Sampling Rate
.....
    
```

Figure 14. Application Decoding OPX File to a Wave File



```

COM47:115200baud - Tera Term VT
File Edit Setup Control Window Help
OPUS Encode and Decode Example...
/> ls
D---- 2016/01/21 11:04      0 test
--A  2016/04/25 11:34 25841760 s_16m_48k.wav
--A  2016/04/25 13:59 2153522  s_8m_8k.wav
D---- 2016/04/25 16:06      0 old

  2 File(s), 27995282 bytes total
  2 Dir(s), 738413778K bytes free

/> enc s_8m_8k.wav s_8m_8k.opx
Encoding 1 channel 8 bits 8000 Hz WAU file
OPUS Encoder Completion: 100
*****STATISTICS*****
Raw WAU Bytes      = 2153478
Encoded OPX Bytes  = 507655
Compression Factor = 04.24

/> dec s_8m_8k.opx out_8m_8k.wav
DEC: Original File is 2153477 bytes
Decoding 1 Channels at 8000 Sampling Rate
.....
*****STATISTICS*****
Input OPX Bytes    = 507655
Decoded WAU Bytes  = 2153600
Creating Wave File

/>
    
```

Figure 15. Application Prints Statistics for the Decompression

4.2 OPUS Audio Playback for OPX

Once the code has been flashed to the TM4C129XNCZAD, you can run the application example as shown below:

1. Press the *Resume* button (or F8) in Code Composer Studio to start execute. The display will shows the welcome screen with the file menu (see [Figure 16](#)).



Figure 16. Application opus_playaudio_opx as Seen on the LCD Panel

2. Select a file with the extension OPX and click on the Play button. The top-right of the screen shows the properties of the file (as read from the Header Segment) and the status bar on the bottom-right displays “Now Playing...” (see [Figure 17](#)). The audio will now be audible on the on-board speaker.

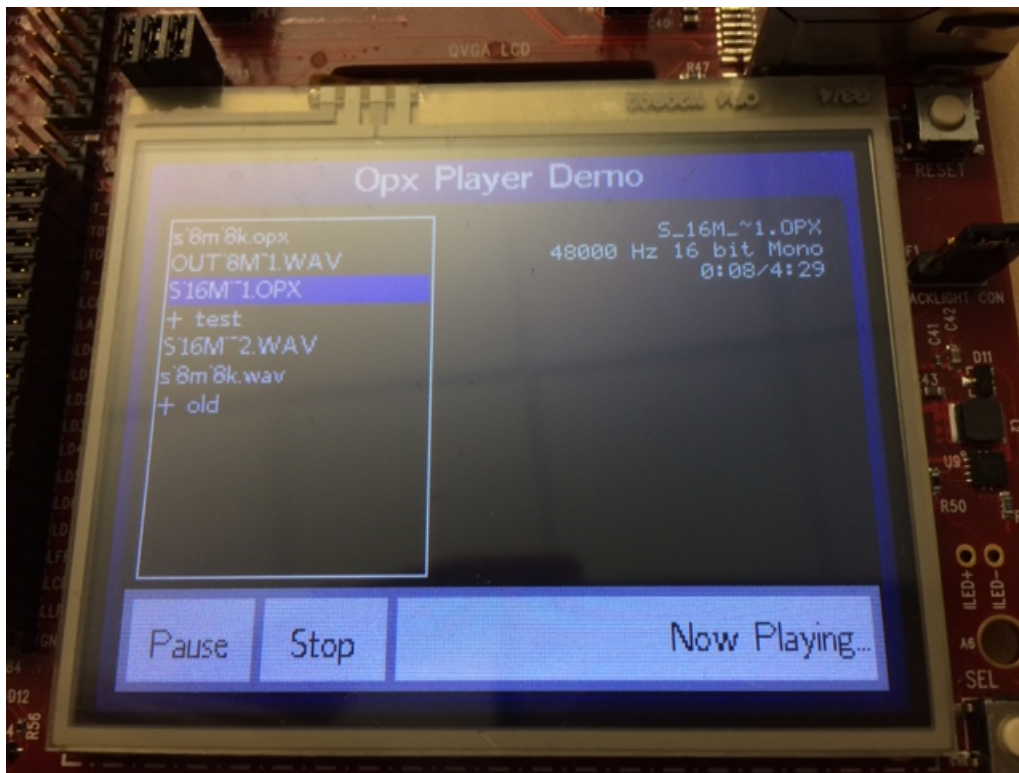


Figure 17. Application opus_playaudio_opx Running the Decoder

4.3 OPUS Audio Playback for OggS-Opus

Once the code has been flashed to the TM4C129XNCZAD, run the application example as shown below:

1. Press the *Resume* button (or F8) in Code Composer Studio to cause the program to start execute. The display will show the welcome screen with the file menu (see [Figure 18](#)).



Figure 18. Application opus_playaudio_ogg as Seen on the LCD Panel

2. Select a file with the extension OPUS and click on the *Play* button. The top-right of the screen will show the properties of the file (as read from the Header Segment) and the status bar on the bottom-right will display *Now Playing...* (see [Figure 19](#)). The audio will now be audible on the on-board speaker.



Figure 19. Application opus_playaudio_ogg Running the Decoder

5 Performance Data on OPUS Encoder

The following two tables illustrate the performance of OPUS encoder on the TM4C129x microcontroller. To enable the performance measurement on the microcontroller add the define `PERFORMANCE_TEST` to the example code `opus_enc_dec` and recompile the project. When the project is compiled with this define the command `testenc <WAVFILE>` is activated. This command loops though the complexity level 0-10 of the OPUS encoder and does not generate any output OPX file.

For all tests, the following parameters are used:

- Output bit rate is set to twice the input wave file sample rate
- Bandwidth is set to `OPUS_AUTO`
- Only CELT mode is used
- The frame size is always 20 ms

To ensure that excessive time is not spent it is advised that the test be done on a wave file not exceeding 10-40 seconds.

The measurements have been performed for 16-bit wave file with sampling rate of 8 KHz (see [Table 1](#)) and 16-bit wave file with sampling rate of 16 KHz (see [Table 2](#)). The method to read the performance data is shown below:

- Column-1 (Complexity): This is the encoder's computational complexity from 0-10 with 10 representing the highest complexity.
- Column-2 (Raw Data Bytes): This is the total number of bytes read from the input wave file.
- Column-3 (Output Data Bytes): This is the total number of bytes output from the encoder.

- Column-4 (Segments): This is total number of frames read from the input wave file. Each segment contains encoder output data corresponding to the frame size.
- Column-5 (Total Time): This is the total time taken for the OPUS encoder to encode the incoming wave file and does not include the time spent in reading the SD card or pre/post processing the data.
- Column-6 (Compression Factor): This is the ratio of compression achieved by the encoder and is computed by dividing Column-2 by Column-3.
- Column-7 (Segment Encoding Time): This is the average time taken by the encoder to encode one frame size worth of input wave file data into the output segment.

NOTE: The playback time of the data stream file can be computed as follows:
 Total Playback time = (Column-2 * Bits per sample) / (8 * Sampling Rate).

Table 1. Opus Encoder Performance for 16-Bit Audio Data Sampled at 8 KHz

Complexity	Raw Data Bytes	Output Data Bytes	Segments	Total Compression Time for wav File (in seconds)	Compression Factor	Segment Encoding Time (in ms)
0	699572	83927	2187	8.825	8.33	4.035
1	699572	83739	2187	9.972	8.35	4.559
2	699572	83885	2187	10.511	8.33	4.806
3	699572	83957	2187	10.570	8.33	4.833
4	699572	83644	2187	10.699	8.36	4.892
5	699572	85617	2187	20.734	8.17	9.480
6	699572	85617	2187	20.734	8.17	9.480
7	699572	85617	2187	20.734	8.17	9.480
8	699572	85617	2187	21.012	8.17	9.607
9	699572	85617	2187	21.012	8.17	9.607
10	699572	85617	2187	21.012	8.17	9.607

Table 2. Opus Encoder Performance for 16-Bit Audio Data Sampled at 16 KHz

Complexity	Raw Data Bytes	Output Data Bytes	Segments	Total Compression Time for wav file (in seconds)	Compression Factor	Segment Encoding Time (in ms)
0	240002	30231	376	1.958	7.93	5.208
1	240002	30187	376	2.171	7.95	5.775
2	240002	30219	376	2.313	7.94	6.152
3	240002	30293	376	2.334	7.92	6.207
4	699572	30269	376	2.362	7.92	6.283
5	699572	30545	376	4.109	7.85	10.929
6	699572	30545	376	4.109	7.85	10.929
7	699572	30545	376	4.109	7.85	10.929
8	699572	30565	376	4.129	7.85	10.984
9	699572	30565	376	4.129	7.85	10.984
10	699572	30565	376	4.130	7.85	10.984

As can be seen from the performance tables, complexity 0 of the OPUS audio codec provides a good compression of the raw wave file, while using 25% of the CPU bandwidth for the codec to run a Cortex M4 core.

There is scope to reduce the computational effort, if the application developer can create a lightweight version of the codec, which is more suitable for low memory and mid performance microcontrollers from TI.

6 Conclusion

In conclusion, the OPUS Audio Codec can be utilized efficiently on the TM4C129x microcontroller to enable low to mid performance applications that require audio to be captured, stored and played back. This is extremely useful in applications such as VoIP-like phones, information kiosks and HMI devices in industrial markets where low-cost exchange or storage of voice data have to be enabled.

7 References

The following related documents and software are available on the [TM4C](#) web page:

- [Opus Interactive Audio Codec](#)
- *Tiva™ DK-TM4C129X Getting Started Guide* ([SPMU361](#))
- *Tiva™ TM4C129XNCZAD Microcontroller Data Sheet* ([SPMS444](#))
- [TivaWare](#) Software
- Ogg Encapsulation Format [RFC3533](#)
- Ogg Encapsulation for the Opus Audio Codec [RFC7845](#)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have **not** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

Audio	www.ti.com/audio
Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
OMAP Applications Processors	www.ti.com/omap
Wireless Connectivity	www.ti.com/wirelessconnectivity

Applications

Automotive and Transportation	www.ti.com/automotive
Communications and Telecom	www.ti.com/communications
Computers and Peripherals	www.ti.com/computers
Consumer Electronics	www.ti.com/consumer-apps
Energy and Lighting	www.ti.com/energy
Industrial	www.ti.com/industrial
Medical	www.ti.com/medical
Security	www.ti.com/security
Space, Avionics and Defense	www.ti.com/space-avionics-defense
Video and Imaging	www.ti.com/video

TI E2E Community

e2e.ti.com